ARMY RESEARCH LABORATORY

Natural Computing: Its Impact on Software Development

Som Karamchetty

ARL-TR-2040 February 2000

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory

Adelphi, MD 20783-1197

ARL-TR-2040 **February 2000**

Natural Computing: Its Impact on Software Development

Som Karamchetty

Information Science and Technology Directorate

Abstract

Many software engineering problems stem, in part, from the need for software designers to understand specialized knowledge domains. Current computer software systems are not capable of representing familiar calculation features such as equations, tables, graphs, procedures, and pictures so that these features assist humans to perform calculations in a natural, intuitive way. This report explains the need for these features to present users with "natural" ways of doing calculations—that is, ways analogous to the paper-based techniques used in the absence of computers. Features presented in this way would make computing more transparent and intuitive. In the *Natural Computing* approach proposed in this report, software tools are first developed and then given to domain specialists to use in their calculation methods, knowledge, and data. As domain knowledge changes and grows, and/or new calculation methods are needed, software developers can add new methods and procedures to the existing methods (or delete old ones) and develop successively enhanced versions of application software for use by both specialists and naive end users. Domain information and knowledge can be captured in electronic books and communicated electronically for further expeditious use. Natural Computing eases application system development and accelerates domain knowledge dissemination, leading to quicker development of further knowledge.

Contents

Figures

Tables

1. Introduction

High life-cycle costs and poor quality are the main problems with current software development. Several methods and practices have been proposed to address these problems, with limited success. One of the more successful approaches is to develop generic tools, such as those for word processing, drawing, and databases; these have resulted in the economical development of software. These tools have become very affordable and are of good quality. In many technical scientific domains, however, in which computers are used to *compute* in the earliest sense of the word—to make calculations—software development is still complex, slow, unreliable, incomplete, and very expensive. The result is that software for literal *computing* is often the least satisfactory type of computer software. In large part, this anomaly arises because software developers and domain specialists are usually different groups.

The approach I propose in this report, called *Natural Computing*, is intended to avoid this problem by providing domain specialists with software tools whose use and function are transparent and intuitive. In the term *Natural Computing*, the word *computing* is thus used in its basic sense, rather than the extended one that has arisen with the development of modern digital computers and particularly personal computers. The word *natural* is used to refer to the kind of understanding of paper-based means of computing that practitioners of technical subjects call on to read and work with technical documents. It could be argued that these techniques are hardly *natural*, since they have been developed over centuries (and individual people must learn them as part of their education and technical apprenticeship); however, I argue that they are *natural* in the same sense that many learned behaviors (such as riding a bicycle) can seem natural to a human being. It is *natural* in the same sense that computer scientists use in referring to the specialized field of *natural language*.

2. Natural Computing Features

Since Natural Computing is based on the way people actually compute using paper-based techniques, the first question to be addressed is "How do people compute—using available information of all types?" Before the invention of computers, most knowledge was captured in the form of books and other paper documents. Books can be further classified as textbooks, reference books, handbooks, and journals, based on the temporal nature of the information. Other documents are flyers, brochures, and receipts, which have a highly transient nature. (I use *books* and *documents* interchangeably to refer to paper-based information.) Information was printed on paper for storage, retrieval, and communication. The paper-based information was read by the end user. By reading the information from one or more documents and by combining it with one's own intuition, invention, and discovery, one generated new information and wrote (printed) it in the form of another paper document.

When we dealt with technical matter, domain knowledge was captured in the form of text that contained equations, tables, graphs,* and pictures.† Without pictures, descriptions of scenes were elaborate (a lot of text). The famous saying "a picture is worth a thousand words" sums it up. Again, in technical subjects, the pictures could be sketches, schematics, drawings, paintings, or photographs. Sketches stood for descriptions of parts and components, such as shapes and sizes. Schematics and other diagrams showed the mutual relationships of components in a system and the state of the system and its temporal nature.

Tables captured relationships among sets of variables. Graphs also represented relationships among variables but additionally provided a highly visual insight into the mutual dependency of the variables.

Domain specialists read the text and concurrently used tables, graphs, and charts. They used a note pad to make temporary notes and calculations. Simple calculations were done mentally. More complicated calculations required aids, such as log tables and slide rules. Domain specialists captured new ideas and information in the form of more equations, tables, graphs, and pictures; appended them to text; and communicated the new documents to others in the field. Documents were subject to three principal types of use: (1) reading and comprehension; (2) interactive calculations using the tables, equations, graphs, and pictures along with the text; and (3) development and recording of new functions (tables, equations, graphs, and pictures). Capturing these essential natural forms and processes in a computer software system is the goal of Natural Computing.

*The term *graph* is used in literature with various meanings. In this paper, a graph means a curve or a set of curves on a graph paper. *Graphs* as in graph theory and *graphics* as in pictures are not the meanings implied.

†The term *picture* is used in this paper to include sketches, schematic diagrams, and drawings. It does not include photographs and painted pictures.

3. A Textbook Example of Natural Computing Features

An example of paper-based computation tools from a textbook illustrates the features proposed for Natural Computing. Figure 1 shows a sample page from an engineering textbook describing mechanical springs. The page consists of a sketch of a mechanical spring, text, and equations. Figure 2 shows another sample page, with a graph, more equations, and more text. By reading the explanations on these pages, an engineer will understand the domain of mechanical springs. An engineer can study the graph on the page and understand the trends. At any time, the engineer can obtain values given by the graphs—this is usually called reading a graph. While using these pages, the engineer starts with values of D and d, proceeds to calculate the value of the variable C from equation 8-1 (a reference number in the sample page (fig. 1)), and reads the value of the Wahl correction factor K from the graph of spring index versus stress correction factor. This value of K and an input value of F (force) are next substituted into equation 8-4 and the value of stress τ is calculated (fig. 2). The engineer may next proceed to the sample page shown in figure 3, and read the values of A and m for a given material. The engineer can substitute these values into equation 8-10 to calculate the ultimate strength in tension of the spring material.

This description shows that domain specialists present information in textbooks for use by others in performing calculations. Thus, textbooks are used both to explain the subject and to provide information in the form of text, sketches, equations, graphs, and tables for ready use in calculations. In the following sections, I describe how traditional computing (using computer software) failed to follow the natural calculations paradigm and is accordingly beset by problems.

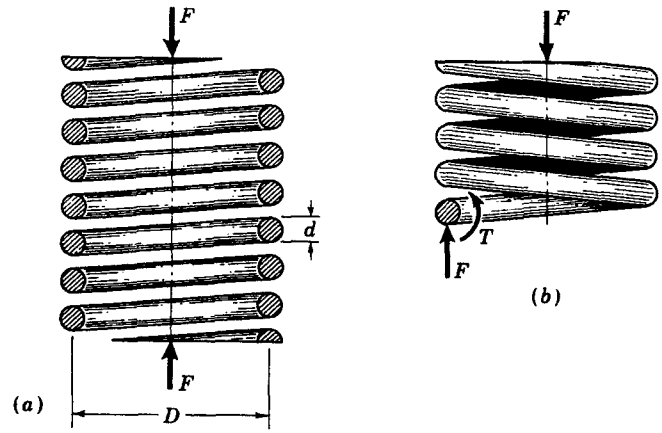


FIGURE 8-1
(*a*) Axially loaded helical spring; (*b*) free-body diagram showing that the wire is subjected to a direct shear and a torsional shear.

the hose in a straight line perpendicular to the plane of the coil. As each turn of hose is pulled off the coil, the hose twists or turns about its own axis. The flexing of a helical spring creates a torsion in the wire in a similar manner.

Using superposition, the maximum stress in the wire may be computed using the equation

$$\tau_{max} = \pm \frac{Tr}{J} + \frac{F}{A} \qquad (a)$$

where the term Tr/J is the torsion formula of Chap. 2. Replacing the terms by $T = FD/2$, $r = d/2$, $J = \pi d^4/32$, and $A = \pi d^2/4$ gives

$$\tau = \frac{8FD}{\pi d^3} + \frac{4F}{\pi d^2} \qquad (b)$$

In this equation the subscript indicating maximum shear stress has been omitted as unnecessary. The positive signs of Eq. (*a*) have been retained, and hence Eq. (*b*) gives the shear stress at the inside fiber of the spring.

Now define *spring index*

$$C = \frac{D}{d} \qquad (8\text{-}1)$$

as a measure of coil curvature. With this relation, Eq. (*b*) can be arranged to give

$$\tau = \frac{8FD}{\pi d^3}\left(1 + \frac{0.5}{C}\right) \qquad (c)$$

Or designating

$$K_s = 1 + \frac{0.5}{C} \qquad (8\text{-}2)$$

Source: Joseph E. Shigley (1977). *Mechanical Engineering Design*, 3rd ed., McGraw-Hill Book Co., New York, NY.

Figure 1. A sample page containing text, sketch, and equations.

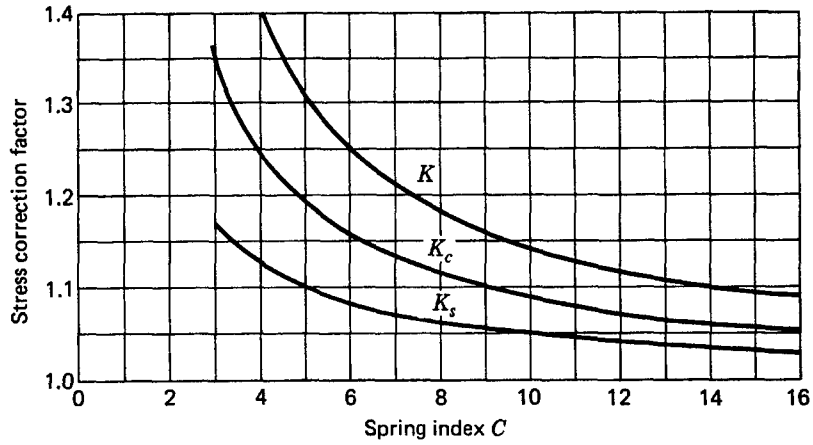


FIGURE 8-2
Values of the stress correction factors for round helical extension or compression springs.

then
$$\tau = K_s \frac{8FD}{\pi d^3} \qquad (8\text{-}3)$$

where K_s is called a *shear-stress multiplication factor*. This factor can be obtained from Fig. 8-2 for the usual values of C. For most springs, C will range from about 6 to 12. Equation (8-3) is quite general and applies for both static and dynamic loads. It gives the maximum shear stress in the wire, and this stress occurs at the inner fiber of the spring.

Many writers present the stress equation as

$$\tau = K \frac{8FD}{\pi d^3} \qquad (8\text{-}4)$$

where K is called the *Wahl correction factor*.* This factor includes the direct shear, together with another effect due to curvature. As shown in Fig. 8-3, curvature of the wire increases the stress on the inside of the spring but decreases it only slightly on the outside. The value of K may be obtained from the equation

$$K = \frac{4C - 1}{4C - 4} + \frac{0.615}{C} \qquad (8\text{-}5)$$

or from Fig. 8-2.

By defining $K = K_c K_s$, where K_c is the effect of curvature alone, we have

$$K_c = \frac{K}{K_s} \qquad (8\text{-}6)$$

Source: Joseph E. Shigley (1977). *Mechanical Engineering Design*, 3rd ed., McGraw-Hill Book Co., New York, NY.

Figure 2. A sample page containing text, graph, and equations.

strengths for various wire sizes and materials.* But the availability of the scientific electronic calculator now makes such a tabulation unnecessary. The reason for this is that a log-log plot of the tensile strengths versus wire diameters is a straight line. The equation of this line can be written in terms of the ordinary logarithms of the strengths and wire diameters. This equation can then be solved to give

$$S_{ut} = \frac{A}{d^m} \qquad (8\text{-}10)$$

where A is a constant related to a strength intercept, and m is the slope of the line on the log-log plot. Of course such an equation is only valid for a limited range of wire sizes. Table 8-2 gives values of m and the constant A for both English and SI units for the materials listed in Table 8-1.

Although the torsional yield strength is needed to design springs, surprisingly, very little information on this property is available. Using an approximate relationship between yield strength and ultimate strength in tension,

$$S_y = 0.75S_{ut} \qquad (8\text{-}11)$$

and then applying the distortion-energy theory gives

$$S_{sy} = 0.577S_y \qquad (8\text{-}12)$$

and provides us with a means of estimating the torsional yield strength S_{sy}. But this method should not be used if experimental data are available; if used, a generous factor of safety should be employed, especially for extension springs, because of the uncertainty involved.

Variations in the wire diameter and in the coil diameter of the spring have an effect on the stress as well as on the spring scale. Large tolerances will result in

* See, for example, the second edition of this book: Joseph E. Shigley, "Mechanical Engineering Design," 2d ed., p. 362, McGraw-Hill Book Company, New York, 1972.

Table 8-2 CONSTANTS FOR USE IN EQ. (8-10) TO ESTIMATE THE TENSILE STRENGTH OF SELECTED SPRING STEELS

Material	Size range, in	Size range, mm	Exponent, m	Constant, A	
				kpsi	MPa
Music wire[a]	0.004–0.250	0.10–6.5	0.146	196	2170
Oil-tempered wire[b]	0.020–0.500	0.50–12	0.186	149	1880
Hard-drawn wire[c]	0.028–0.500	0.70–12	0.192	136	1750
Chrome vanadium[d]	0.032–0.437	0.80–12	0.167	169	2000
Chrome silicon[e]	0.063–0.375	1.6–10	0.112	202	2000

[a] Surface is smooth, free from defects, and with a bright lustrous finish.
[b] Has a slight heat-treating scale which must be removed before plating.
[c] Surface is smooth and bright, with no visible marks.
[d] Aircraft-quality tempered wire; can also be obtained annealed.
[e] Tempered to Rockwell C49 but may also be obtained untempered.

Source: Joseph E. Shigley (1977). *Mechanical Engineering Design*, 3rd ed., McGraw-Hill Book Co., New York, NY.

Figure 3. A sample page containing text, table, and equations.

4. Problems With Traditional Software Development

While discussing the problems plaguing the software industry, Coad and Yourdon (1991) state, "systems analysts must first understand the problem domain at hand." They also answer the questions "What is so difficult about analysis? And what is the challenge?" by identifying four major difficulties faced by systems analysts on all types of projects: problem domain understanding, person-to-person communication, continual change, and reuse:

- *Understanding.* Scientists devote their lifetimes to understanding their specific domains and to enriching them by discovering new knowledge. No wonder software systems analysts find it impossible to "understand" the domain.
- *Communication.* In any domain, different practitioners hold differing viewpoints and preconceptions that make communications among them difficult. Needless to say, communication with outsiders (such as systems analysts) seems well-nigh impossible.
- *Change.* As scientists work in a specific area, they hit upon new ideas constantly—that is their goal. Therefore, it is a *given* that a domain will continually change. (Modeling a domain that has ceased growing is easier for the systems analyst, of course, but the result may not be useful.)
- *Reuse.* Modern economies depend on the reuse of all knowledge and tools so that the many may enjoy the fruits of the labor (and inventiveness) of the few. Bricks and bolts and nuts are probably the best examples of engineering designs of great reuse.

One can avoid these four types of difficulties, which are inherent in conventional software development, by using Natural Computing. As I conceive it in Natural Computing—

- Software developers do not need to understand a whole domain: they need only provide building blocks for the specialist to work with.
- Communication between domain specialists and software developers at the domain level of abstraction is unnecessary.
- The domain will be extended by the scientist (or engineer or domain analyst), not the software developer.
- Software engineers are provided ways to program building blocks.

4.1 Development of Tools Versus Development of Domain Knowledge

To assist a discipline, software developers do not need to grasp all the knowledge of that discipline, but they must understand its language and medium. Every field has developed notations, techniques, and methods for facilitating communications among practitioners within and between fields. Scientists and their followers constantly reuse these forms of

knowledge for further understanding and for applications that benefit society. Both the methods of mathematicians and the tools and techniques of engineers provide such benefits.

Certain disciplines, such as mathematics, enable other disciplines to deliver goods to society. All sciences depend on mathematics to some degree to solve problems that can be formulated in mathematical terminology. Mathematicians do not, however, insist on "understanding" a science domain. Consider, for example, the application of quadratic equations. A scientist does not explain a domain problem to a mathematician so that a quadratic equation solution can be applied. Instead, mathematicians have suggested and supplied a number of methods for solving quadratic equations without regard to their particular application. In general, mathematicians have provided scientists with a vast array of theories and methods (such as complex variables, Laplace transforms, Fourier transforms, and Bessel functions). Following the example of the mathematicians, software developers and software engineers should abstain from demanding a full understanding of the domain in which they are developing applications. They will be able to contribute to human progress by providing free-standing methods for use by scientists and engineers. Software engineering professionals should develop the tools and enable the domain specialists to incorporate the domain knowledge and methods with the help of such tools. In this way, software is like mathematics, and its usefulness to humanity can be realized only through other domains.

As scientists discover new facts, they codify their knowledge and present it in papers, articles, books, and handbooks. This knowledge is in the form of text, equations, tables, graphs, procedures, and pictures. These are the essential forms of the language of science and technology (as well as of other domains in which calculations play a part, such as engineering, accounting, economics, business management, and (even) political science). Natural Computing focuses on these forms, so that software engineers can develop tools that deal with computer representations of these forms.

4.2 Generic Software

Some examples of generic software development can be cited. Word processing, music, and graphics programs are available that are independent of a specific domain. Because these software tools employ the users' language and notation and emulate their familiar tools, they can be used by a variety of practitioners. To the credit of such generic software, musicians can use computers to create music, and painters can use computers to paint. Software developers, in general, are likely to agree that it is best to assist artists, musicians, and painters by giving them appropriate tools, rather than trying to understand these richly complex domains. Unfortunately, however, when it comes to scientific, engineering, and analysis domains, developers attempt to *understand* the domain and develop applications programming. This is the crux of the problem in software.

Brooks (1987) argued persuasively that there are no magic solutions to the fundamentally difficult problems associated with software development. There are no panaceas—no miracle cures that will automatically increase our productivity by orders of magnitude while eliminating all the software bugs. However, by focusing on the development of generic tools, software developers can reduce software bugs, and software development can be moved to a higher plane.

4.3 Ends and Means

Coad and Yourdon (1991) caution software practitioners that if the application of a software engineering method produces a monument of paper, then something is wrong—in the method, in the application of the method, or perhaps in both. They lament, "if we lose sight of people and begin producing charts, diagrams, and piles of paper as *ends* [italics added] unto themselves, we fail to effectively communicate." Currently, most software methods generate displays in the form of plots or tables that can be used as records but not as dynamic relationships. Contrast this with a scientist's intuitive and interactive use of tables in books. A scientist can readily use a table on paper either to read or as a relation in his or her computing. Since computer tools are not now available that treat tables as relationships, their reuse by other software is limited and circuitous—reuse depends, in fact, on the scientist's intervention based on understanding—just as with a paper table. In a similar manner, some programs can generate graphs in vivid colors and multiple dimensions and animations (see fig. 4, generated by Mathematica® (Wolfram, 1991)).

In[4]:=

```
Plot3D[Sin[x] Sin[y], {x, 0, 2Pi}, {y, 0, 2Pi}]
```

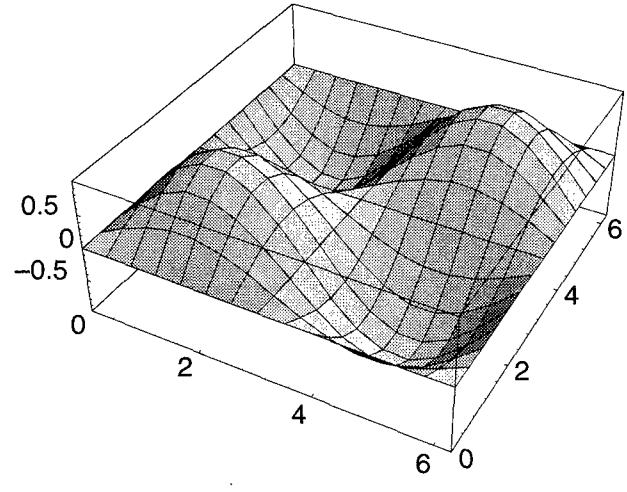


Figure 4. A three-dimensional plot by Mathematica®.

But alas! Neither this graph nor any of the graphs on a computer screen are meant to be used by another program.

I argue that text, equations, tables, graphs, pictures, and other forms of output (*ends*) generated by current programs should actually be *means* for communication among people, understanding by people, and further generation of knowledge by people. The greatest unmet need in software engineering is developing methods and tools that will capture equations, tables, graphs, procedures, and pictures as reusable software objects. Moreover, these forms should be part and parcel of text (computer documents), just as they are in current paper-based texts. This idea is at the heart of Natural Computing.

5. Information Representations and Domains

Equations, graphs, and tables are all intended to capture and represent functional relationships among a set of variables in a given domain. In the following paragraphs, I present examples from widely different domains to stress the point that these three forms can represent relationships in those domains. First, using a scientific example, I consider the relationship among the temperature, pressure, and volume of a gas. In the simplest case, this relationship among the state variables (temperature, pressure, and volume) is represented by an ideal gas equation (fig. 5). The same relationship can be represented by a table (table 1) or by a graph (fig. 6). Any one of the three forms can be used as a means to obtain one value of a state variable, given values of the other two.

Figure 5. Ideal gas equation with notation.

Ideal gas equation:

$V = R \times T/p$

Notation:
V = Volume, ft^3
R = Gas constant, ft-lbf/lbm-deg R
= 53.35 for air
T = Temperature, deg R
p = Pressure, psia

Table 1. Tabular representation of temperature-volume-pressure relationship for an ideal gas.

Table 1. Specific volume of air (ft^3).

		Pressure psia				
		14.7	20	30	50	60
Temperature deg R	529.7	13.35010157	9.812325	6.54155	4.906162	3.92493
	540.0	13.60969388	10.00313	6.66875	5.001563	4.00125
	550.0	13.86172525	10.18837	6.792245	5.094184	4.075347
	560.0	14.11375661	10.37361	6.915741	5.186806	4.149444
	570.0	14.36578798	10.55885	7.039236	5.279427	4.223542
	580.0	14.61781935	10.7441	7.162731	5.372049	4.297639
	590.0	14.86985072	10.92934	7.286227	5.46467	4.371736
	600.0	15.12188209	11.11458	7.409722	5.557292	4.445833

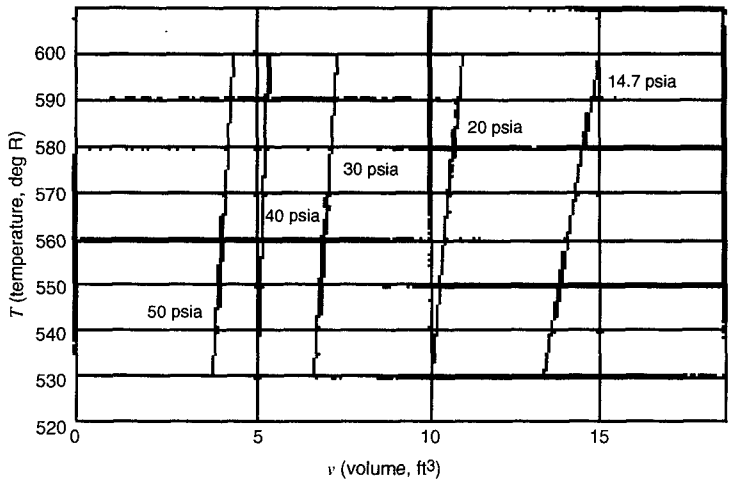


Figure 6. Graphic representation of temperature-volume-pressure relationship for an ideal gas.

Let us explore the genesis of the three forms for representing relationships. In general, mathematical analysis of a problem results in a functional relationship in the form of an equation that we can use in further calculations. Experimentally observed data are set up in a tabular form, or the results are plotted in the form of a graph. These three forms of representation have strong and weak points. Books on data analysis describe such issues in detail, which are beyond the scope of the present discussion. I limit myself to a few remarks. Graphs show the highest visibility of trends. Tables provide some sense for trends but readily reveal magnitudes. Equations are compact, but provide little or no sense of the trend (except in simple cases). Barring such specific virtues, these three forms are identical in capturing relationships between variables. In comparison with these three forms, a black box computer program, which can also embody the relationship among the variables, has the least visibility when it comes to revealing the trend of a function or algorithm (see fig. 7).

As an example of states of a system and information about the system, consider a road map. It represents the states, which are map locations. Roads or paths connect the locations. For an automobile to move from location A to location B, it needs to take a certain path. A map helps in determining which alternative path(s) will take an automobile from A to B. A map is two-dimensional and describes a location in terms of its x and y coordinates. If we superimpose a map of railroads on our (road) map example, the user has more alternative paths to take from A to B. We can provide descriptive directions to a traveler on how to go from A to B, or we can give the traveler a map and let him or her choose a path. Giving directions is algorithmic and brief. On the other hand, by supplying a map, we have provided the traveler with considerable information and insight. Now, the traveler can choose the shortest path, find a scenic route, or replan the route if a roadblock occurs during travel. A map is actually a complex graph. Figures 8 and 9 present examples of a map and algorithmic directions, respectively.

Figure 7. A C++ program for ideal gas equation.

```
main ()
{
real R, T, p, v;
cin >> R;
cin >> T, p;
v = R * T/p;
cout << v;
}
```

Figure 8. A map of a city provides navigational information to a traveler.

Source: AAA.

Figure 9. Algorithmic description of a route.

```
From: Sacramento Met Airport
Take: 5 South
Exit to Rte 80 east
Exit to Rte 80 Bus South
Exit to Exposition Blvd
To: Cal Exposition
```

Relationships connect the various states of a system, and they play a key role in the description of a system as they describe the behavior of a system. Consider the game of baseball. Table 2 shows the performance of the New York Yankees on a particular day. Figure 10 shows the state of the American League and the National League on that day (*The Washington Post*, 1996). As each ball is played, the state of the game between two opposing teams changes. A batter, a pitcher, and other players have attributes, and they change with each pitch thrown. Numerous states are recorded by official scorers and by baseball connoisseurs and aficionados. Table 2 and the tables in figure 10 are one set of abstractions or representations of all possible and generatable states.

Groups of players belong to a team, and the states of the teams (league) are represented in the table in figure 10. When the results of the next day's games are available, this table can be recalculated according to a set of relationships. Take, for example, the following generic equation:

$$\text{new status} = \text{old status} + \text{new result}.$$

For each team,

$W = W + 1$ if the next game was won by this team, or

$W = W$ if the next game was lost or postponed;

$L = L + 1$ if the next game played was lost by this team, or

$L = L$ if the next game played was won or postponed;

$\text{Pct} = W/(W + L)$; and so on.

This example illustrates that a system has many states and that a given abstraction captures and shows selected states. States comprise sets of variables and are connected by means of relationships.

Table 2. Performance status of Yankees on a particular day.

New York	AB	R	H	BI	BB	SO	Avg
Boggs 3b	4	0	1	1	0	0	0.331
Girardi c	4	0	0	0	0	0	0.282
O'Neill rf	3	0	1	0	1	0	0.368
TMartinez 1b	4	0	0	0	0	0	0.246
Sierra dh	3	2	3	0	1	0	0.281
Duncan 3b	4	0	0	0	0	0	0.343
GeWilliams lf	3	0	0	1	0	0	0.333
RRivera cf	2	0	1	1	1	0	0.5
BeWilliams ph	1	0	0	0	0	1	0.278
DJeter ss	3	1	1	0	0	0	0.275
Totals	31	3	7	3	3	1	—

Finally, I present a technical example, the case of a thermal system. The operation of a steam plant is shown in figure 11. There are four components: a boiler, a turbine, a condenser, and a pump. The states of the working fluid (steam) are represented on a temperature-entropy (T-s) diagram (Van Wylen and Sonntag, 1965). At a given state, the properties (attributes) of the steam of interest to a thermal engineer are p, T, v, s, and h. Specific relationships apply to the components that affect a change in

Figure 10. Status of American and National leagues.

BASEBALL

AMERICAN LEAGUE STANDINGS

EAST	W	L	Pct	GB	L10	Strk	Home	Away
New York	27	19	.587	—	5-5	L1	18-8	9-11
Baltimore	27	20	.574	½	7-3	W1	18-11	9-9
Toronto	21	28	.429	7½	3-7	L3	10-13	11-15
Boston	19	28	.404	8½	6-4	L1	13-12	6-16
Detroit	12	38	.240	17	0-10	L11	6-17	6-21

CENTRAL	W	L	Pct	GB	L10	Strk	Home	Away
Cleveland	33	14	.702	—	8-2	W4	16-5	17-9
Chicago	29	18	.617	4	9-1	W8	16-5	13-13
Milwaukee	22	25	.468	11	5-5	L4	10-10	12-15
Minnesota	22	26	.458	11½	4-6	W3	12-13	10-13
Kansas City	23	28	.451	12	6-4	L2	10-15	13-13

WEST	W	L	Pct	GB	L10	Strk	Home	Away
Texas	30	19	.612	—	4-6	W2	18-7	12-12
Seattle	26	22	.542	3½	6-4	W1	15-12	11-10
California	23	25	.479	6½	3-7	W1	15-9	8-16
Oakland	22	26	.458	7½	3-7	L1	10-12	12-14

SUNDAY'S RESULTS

- Baltimore 6 ..... Oakland 1
- Minnesota 9 ..... Toronto 3
- Cleveland 5 ..... Detroit 0
- Chicago 12 ..... Milwaukee 1
- Texas 6 ..... Kansas City 4
- at California 12 ..... Boston 2
- at Seattle 4 ..... New York 3

MONDAY'S GAMES

- Boston (Wakefield 2-5) at Oakland (Wojciechowski 5-0) 4:05
- Chicago (Fernandez 5-2) at Toronto (Janzen 2-0) 7:35
- Detroit (Gohr 2-6) at Kansas City (Belcher 5-2) 8:05
- Cleveland (McDowell 5-2) at Texas (Pavlik 6-1) 8:35
- New York (Pettitte 6-3) at California (Abbott 1-7) 10:05

TUESDAY'S GAMES

- Baltimore at Seattle 10:05
- Chicago at Toronto 7:35
- Minnesota at Milwaukee 8:05
- Cleveland at Texas 8:35
- New York at California 10:05
- Boston at Oakland 10:05

SATURDAY'S RESULTS

- Oakland 6 ..... Baltimore 3
- Minnesota 6 ..... Toronto 4 (10)
- Cleveland 7 ..... Detroit 6
- Chicago 9 ..... Milwaukee 7
- Texas 2 ..... Kansas City 1
- Boston 10 ..... California 3
- New York 5 ..... Seattle 4

NATIONAL LEAGUE STANDINGS

EAST	W	L	Pct	GB	L10	Strk	Home	Away
Atlanta	32	17	.653	—	7-3	W1	20-9	12-8
Montreal	29	21	.580	3½	2-8	L3	16-7	13-14
Philadelphia	24	24	.500	7½	4-6	W1	8-11	16-13
Florida	25	26	.490	8	5-5	W1	16-11	9-15
New York	20	28	.417	11½	5-5	W1	10-13	10-15

CENTRAL	W	L	Pct	GB	L10	Strk	Home	Away
Houston	25	26	.490	—	5-5	W3	12-14	13-12
St. Louis	22	27	.449	2	6-4	L1	9-12	13-15
Cincinnati	19	25	.432	2½	3-7	L1	10-13	9-12
Chicago	21	29	.420	3½	3-7	L4	15-12	6-17
Pittsburgh	19	30	.388	5	3-7	L1	8-16	11-14

WEST	W	L	Pct	GB	L10	Strk	Home	Away
San Diego	31	19	.620	—	6-4	L1	17-11	14-8
Los Angeles	27	24	.529	4½	7-3	W3	16-9	11-15
San Francisco	25	23	.521	5	5-5	L1	11-13	14-10
Colorado	23	23	.500	6	8-2	W2	15-10	8-13

SUNDAY'S RESULTS

- Los Angeles 4 ..... Montreal 3
- Florida 8 ..... St. Louis 2 (7, rain)
- Atlanta 6 ..... at Pittsburgh 3 (13)
- New York 1 ..... San Diego 0
- Houston 7 ..... Chicago 2
- Philadelphia 10 ..... at San Francisco 1
- Cincinnati ..... at Colorado (ppd, rain)

MONDAY'S GAMES

- Houston (Kile 5-3) at Pittsburgh (Ruebel 0-0) 1:05
- Colorado (Reed 4-4) at St. Louis (Stottlemyre 4-2) 1:15
- Atlanta (Glavine 5-3) at Chicago (Telemaco 2-0) 4:05
- Cincinnati (Smiley 4-4) at Florida (Leiter 6-4) 4:35
- San Diego (Hamilton 7-3) at Montreal (Fassero 3-4) 7:35

TUESDAY'S GAMES

- Cincinnati at Florida 7:05
- Houston at Pittsburgh 7:05
- San Diego at Montreal 7:35
- Los Angeles at Philadelphia 7:35
- San Francisco at New York 7:40
- Atlanta at Chicago 8:05
- Colorado at St. Louis 8:05

SATURDAY'S RESULTS

- San Diego 7 ..... New York 2
- San Francisco 3 ..... Philadelphia 2
- Pittsburgh 5 ..... Atlanta 2
- St. Louis 5 ..... Florida 0
- Los Angeles 5 ..... Montreal 3
- Houston 5 ..... Chicago 2
- Colorado 7 ..... Cincinnati 5

Source: *The Washington Post*, Monday, 27 May 1996, p C6.

Figure 11. Simple steam power plant schematic and temperature-entropy diagram.

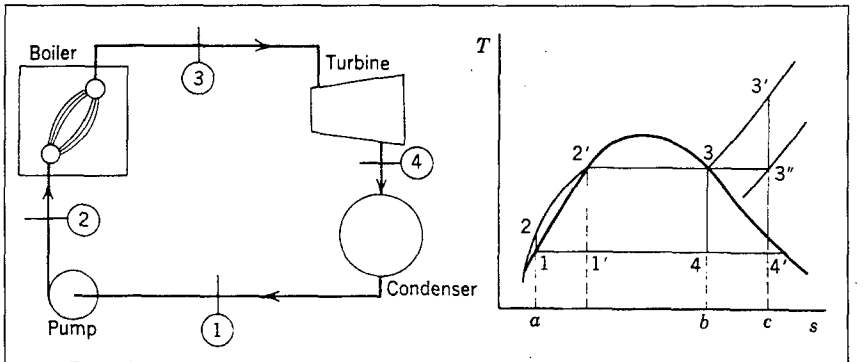
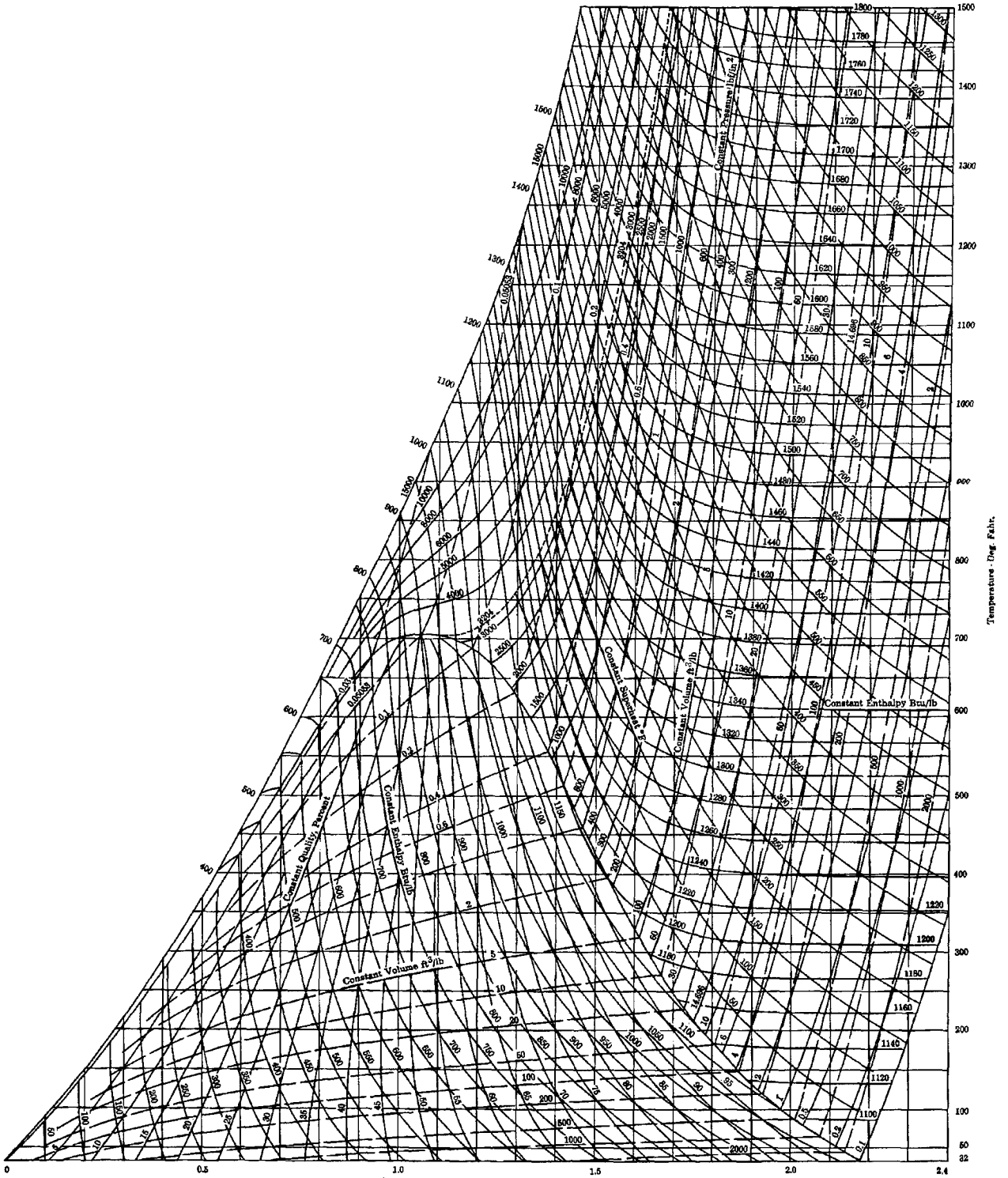


Fig. 9.1 Simple steam power plant which operates on the Rankine cycle.

Source: Gordon J. Van Wylen and Richard E. Sonntag (1965). *Fundamentals of Classical Thermodynamics*, John Wiley and Sons, Inc., New York, NY.

the state of the steam. Expressed in other words, specific paths connect the states (1–2, 2–3, 3–4, and 4–1). Historically, the calculation of the states (i.e., the values of the properties) was done by the use of steam tables and steam charts, called Mollier diagrams (fig. 12) (Keenan et al, 1969). Mollier charts are highly visual, and their use made the "physics" of the steam plant process highly intuitive. However, charts are not accurate, since they are limited by their scale. The use of steam tables improved the accuracy of the calculations, albeit at the expense of some degree of visibility. Since the advent of digital computers, equations were fitted (or derived) for the behavior of steam. The equations are extremely complex, and their solution by computer algorithms is a "black box" process.

In every domain, information is essentially captured and represented in the form of equations, tables, and graphs. As discussed previously, traditional computer programs catered only for algorithmic information. However, since people need both information and algorithms to understand the principles involved in the domain and to perform calculations, a scheme where both information and algorithms can simultaneously be represented and presented has great merit and utility.



Source: Joseph H. Keenan, Frederick G. Keyes, Philip G. Hill, and Joan G. Moore (1969). *Steam Tables: Thermodynamic Properties of Water Including Vapor, Liquid, and Solid Phases*, John Wiley and Sons, Inc., New York, NY.

Figure 12. Properties of steam represented on a Mollier chart.

6. Ideal Characteristics of Software Tools

Having described the features of paper-based computing that people have evolved over time, I am ready to state some goals for Natural Computing. The main goal of good software tools should be to provide a user with facilities to read, interactively calculate (as on a scratch pad), and develop and record new procedures and information in any calculation-intensive domain. Once such good tools are successfully developed, electronic information processing can be cleanly divided into three distinct parts: (1) software tool development, (2) domain setup and knowledge incorporation, and (3) end use. In such a scenario, software developers create generic tools that know how to operate on text, equations, tables, graphs, and pictures. Domain specialists set up and incorporate data, information, and procedures, using the generic tools provided to them by software developers. Finally, end users use the domain information and procedures provided to them. There will be a clean and clear-cut division between the software tool development and the domain incorporation. Figure 13 shows this ideal, in contrast with figure 14, which shows what is currently standard. In time, as newer versions of software tools with greater capabilities evolve, digital computing will approach what is natural to people.

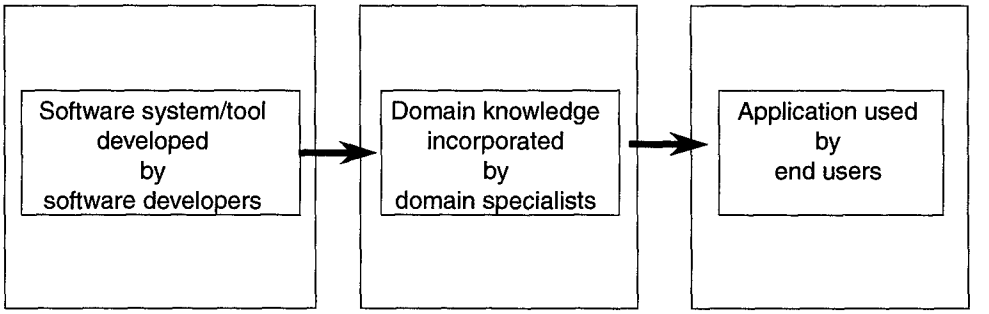


Figure 13. Chronological sequence in ideal Natural Computing.

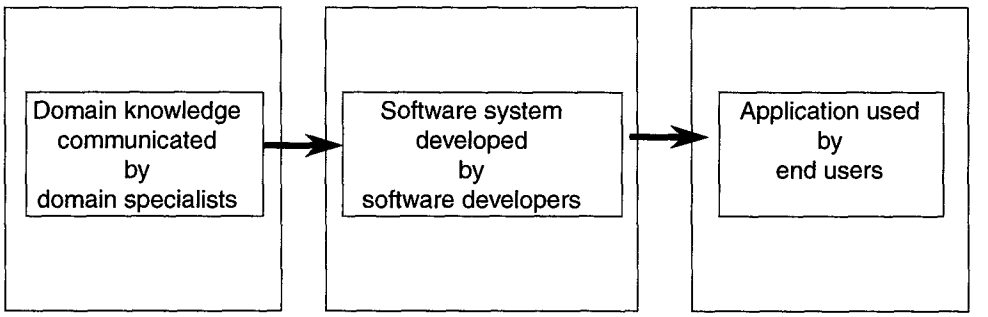


Figure 14. Chronological sequence in current standards.

7. Evolution of Computing

Having proposed an ideal set of characteristics for computing, I now present a quick survey of the state of the art. Approaches to computer-based calculations have continuously evolved. During the fifties and sixties, computers were seen as productivity machines. The nature of software tools or languages was such that domain specialists could use software tools to speed up their calculations and also to apply them to more complex problems. It was found to be more effective to train scientists and engineers in FORTRAN programming than to train software developers in science. Similarly, finance and accounting specialists adopted COBOL programming. Note that software developers did not need to "understand" the domain, because the domain specialists knew the programming languages. By the seventies, however, as larger computer systems became available, these approaches by themselves were not adequate. Rather than providing appropriately more natural FORTRAN and COBOL, software developers became *domain* software developers. The common term for them is "application developers."

Till recently, text, graphs, equations, and pictures did not coexist in computer documents. The eighties saw the development of desktop publishing systems that helped us create excellent documents that included all these various forms. Even so, these objects are generated as *ends* and not as *means* for further continual computing. That is, they are representations of information (just like traditional paper documents), but they are not tools for digital computing.

In the eighties, a number of generic tools emerged that permitted scientists and others to work in symbolic mathematics: Macsyma, SMP, Reduce, MathCAD, TK!Solver, and Mathematica®. The popularity and widespread use of such tools demonstrate that domain specialists want better and more natural tools and not the personal services of "application" developers.

Graphics programming tools have shown that points, lines, surfaces, and so forth, are the language features of a variety of graphics domains, whether for building plans, plumbing, machine designs, or integrated circuits. The graphics programming area has taken giant steps in developing natural software tools that graphic artists can use. Such examples should similarly guide us in recognizing the basic features that are generic to calculations, especially while one is using graphs and pictures.

If we follow this trend into software for calculations, operations on tables, equations, and graphs can be generic or domain-independent. The first step to progress in this area is to represent computing features (tables, graphs, equations, etc) as objects, through object-oriented programming. The next step is to use those features in a variety of domains.

8. State of the Art of Tables in Software

Since I am arguing for an appropriate representation for tables, graphs, equations, and other objects, I first examine the state of the art in computing with respect to these objects, beginning with tables. Tables are now used mainly in two principal areas: database systems and text processing.

8.1 Tables in Database Systems

Date (1995, p 79) calls a table a relation, and then (p 80) states, "a relation and a table are not really the same thing, although in practice it is frequently convenient to pretend that they are." The vast literature on database systems might lead an unwary reader to conclude that the database community has already represented tables in a computer-usable format. However, database relations do not allow people to use the most common tables that they know. This situation is due to the mathematical rigor of the database systems. Software developers are caught in a dilemma between mathematical rigor and the natural but highly flexible forms people use. For example, study the baseball tables in figure 10 and table 2. Six-year-old children grasp the nuances of these tables. These simple-looking tables represent many relationships. Lay readers can quickly compare and calculate desired outcomes. As new baseball games are played each day, new tables of values can be calculated from new data, old table values, and predefined relationships (formulas). Databases are not meant to be used that way.

Shaler et al (1988) discuss formalizing the concept of a table, including normalization rules for producing well-formed tables. According to Shaler et al, the normalization rules can be viewed from two perspectives. The first focuses on the form of data in databases; these rules tell how to set up tables so that there is little redundancy in the data—that is, the amount of data required to store a certain information content is minimized. The second perspective (the one most *natural* to us) looks at the normalization rules as statements about the repertoire of forms that we use in our model (the fact that we are using tables, for example), and the meaning we imply whenever we use a form in a particular manner.

Date (1995) describes a database system as basically a computerized "record-keeping" system. Its overall purpose is to maintain information and to make that information available on demand. Date considers three classes of users: the application programmers, the end users, and the database administrators. Date also gives a slightly more precise definition of the *database*: "A database consists of some collection of persistent data that is used by application systems of some given enterprise."

While defining what a relational system is, Date states that in the relational system, the user perceives the data as tables (and nothing but tables) (p 22). He goes on to state, "For most practical purposes, indeed, the terms *relation* and *table* can be taken to be synonymous." The relational model is a way of looking at data—that is, it is a prescription for a way of representing data (namely, by means of tables).

A scrutiny of table 2 and the tables in figure 10 will reveal the simple-appearing means used by natural tables to represent a variety of complex relationships; by contrast, the best database systems are much more limited. For example, in natural tables, ranges are represented, blanks are allowed with definite meanings, and data types are mixed with no problems. In the first column of table 2, notice how a batter and information on his field position are combined! A database system will not allow such mixing.

Despite the extensive use and discussion of "tables" in database systems, these entities are much more limited than real-world tables and do not provide a way to capture the complex relationships that *natural* tables embody.

8.2 Tables in Text-Processing Tools

As suggested earlier, tables in word processors and publishing tools are displays only; the meaning relationships among the elements are supplied by the reader and are not inherent in the tables themselves. Discussions of tables in text processing accordingly focus on their display features.

Morris (1996, p 79) describes how tables are formatted and added as a new feature of Hypertext Mark-Up Language (HTML) so that HTML can become a true publishing medium. As is common with HTML documents, tags are used to define a table and its components. A table is divided into rows and cells. Techniques are defined to format text in table cells. However, these formatting rules are merely to represent tables for display. Creation and editing are permitted, but no other data manipulation and use are possible at present.

Lemay (1996, pp 194, 440) discusses formatting of tables for use on the World Wide Web, and again the emphasis is on creating tables at the transmitter's terminal (or server) and their display at the receivers' (client) terminals. To add tables to your web page, Netscape features table heading cells and table data cells. Lemay also suggests the use of lists, images, and preformatted text as alternatives to tables.

In its "Autoformat" feature, Microsoft Word (version 7.0, 1995) accommodates 38 different formats for a table. It also allows for columns to be split (table 3). However, the purpose of these operations is display in a natural-looking fashion. But no methods exist that support the placement, retrieval, or manipulation of data.

As this brief discussion shows, although modern text-processing tools provide more or less sophisticated methods for creating tables as displays, they provide no tools for capturing the relationships represented by tables. Just like paper tables, these (text processing) tables rely on a human interpreter to supply meaning.

Table 3. A table in a word processor showing an element spanning two columns.

Monday	Tuesday	Wednesday	Thursday	
			AM	PM
—	—	—	—	—
—	—	—	—	—
—	—	—	—	—
—	—	—	—	—

8.3 Table Usage in Real World

In the examples of tables given in section 5, I chose the baseball tables to illustrate the ubiquity of tables in the real world. A four- to six-year-old (in U.S.) understands a baseball table. A child of four understands a table of menus and prices at a fast-food restaurant. With only a basic knowledge, the child can easily process this table. Humans gain such processing knowledge independent of the domain. It appears that, initially, they apply the processing knowledge to simple cases in their favorite domains and later extend it to complex domains. As people grow, they comprehend more complex table constructions. With a little help, or through exploration, they understand tables of increasing complexity. And they continually add more complex tables to their repertoire. That is the nature of our learning!

A teenage student may have problems with school homework but understands tables comparing automobiles in the April issue of *Consumer Reports*. These tables do not even use numbers in the cells. Circles filled with red and black colors are used, and footnotes explain what the various colored circles mean.

On a more advanced level, the example of logarithmic tables demonstrates that a table can be much more than the record-keeping tool of database programming. Before the slide rule, the calculator, and the computer led to their demise, log tables were used by scientists and engineers as an integral part of a calculation. Each table element entry in the body of a log table is the value of the logarithm of a given number indicated by a combination of row and column heading. (This convention alone is sufficient to show that a table is not merely a database.) A complementary table contains antilogarithms. Other examples of tables used directly to facilitate computations are tables of values of sine, cosine, tangent, sinh, cosinh, tanh, and so forth.

Of course, those well versed in mathematics preferred to use equations rather than resorting to tables, and still others represented the functional relationship in a graphical form—a curve. This is an instance of the same functional relationship among variables being represented by tables, equations, and graphs. In other instances, such as experimental data and empirical observations (e.g., baseball results), where equations are difficult to fit, tables of data are the only recourse.

How do we use tables? As a relation is the simple answer—or as a repository. We use tables to obtain textual information, symbolic information, graphic information, and finally numerical data. But most of all, we interpret the information for subsequent and continual use. Our software versions of tables should allow us to do the same things.

9. An Analysis of Tables

Having described in the previous section how tables are represented in traditional databases and text-processing documents, I analyze in this section some characteristics of tables as they are used in real life, describing their anatomy and morphology; I suggest that these characteristics should be captured in software, so as to allow software operations on tables that parallel the real-world operations that scientists and engineers perform with paper tables.

A body of knowledge has developed about tables, their representation, and their behavior. This knowledge is essentially empirical and unwritten. When we try to program the structure and behavior of tables in software, it is important to understand and capture these traditional conventions and nuances.

9.1 Anatomy and Morphology

9.1.1 Table Parts

Figure 15 shows the terminology for parts of a table used in printing by the U.S. Government Printing Office (GPO, 1984). However, since such terminology is not inclusive (and much of it is related strictly to the display aspect of tables), I depart from it and use my own terms for certain parts of a table.

A table represents some characteristics and values in a domain (such as the standings of a baseball team). Here I start with a two-dimensional rectangular array as a common example of a table, while not restricting my discussion to it. As shown in figure 16, a table can be divided into the title, the set of column labels, and the table body; further, within the body, the first column often has special status.

The *table caption* gives it an identity, describing the subject matter of the table. In documents such as books, we find lists of tables that bring together all the table titles to one location. The list of titles (including page numbers) is called the table of contents and is used as an index into the book.

I refer to the set of column labels as the *column header*, which is called in GPO terms the "boxhead." This part of the table describes the characteristics that are represented in the columns. Table column headers (boxheads) are usually complex data forms. For example, the elements may carry a characteristic, an abbreviated variable or symbol standing for the characteristic, and appropriate units for the characteristic—such as "Frequency, f (Hz)." It is often not sufficient to represent a characteristic alone; for practical utility, its units must also be included. User-friendly, practical tables show the units for a given characteristic in the same or in an adjoining element. Figure 17 shows the anatomy of a table column header (boxhead).

DEFINITION AND PARTS OF A TABLE

To define and describe fully all of the many parts, terms, and details which enter into tabular presentation is difficult to explain in a few words or to understand readily without an accompanying visual example. The example shown is directed at those concerned with the construction and makeup of tables, with guidelines identifying tabular terms and details. Many of the terms can be applied to any form of tabular matter.

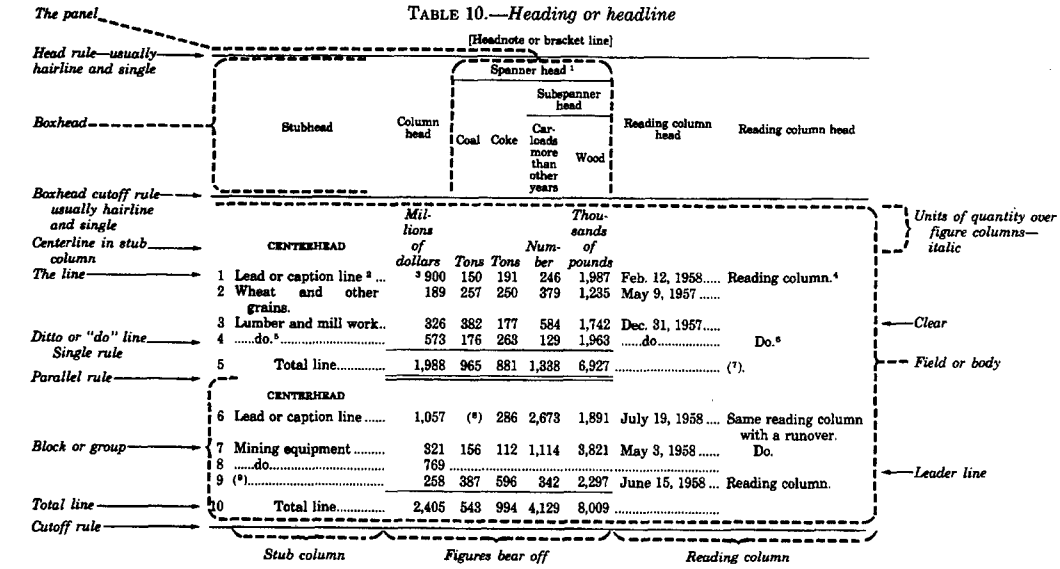
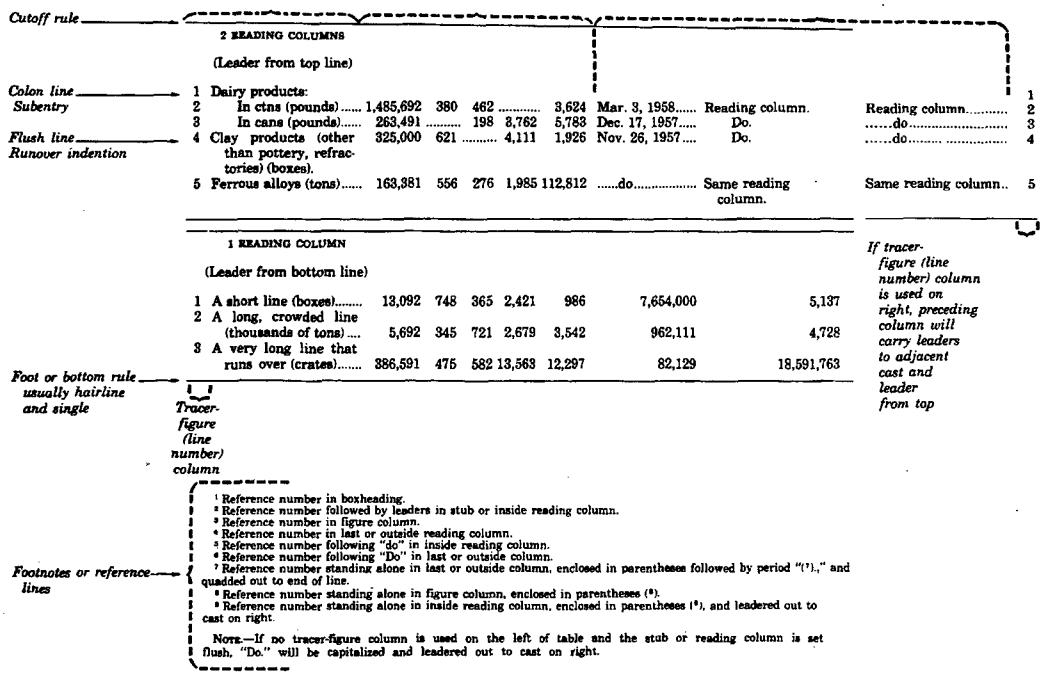


TABLE 10.—*Heading or headline*

[Headnote or bracket line]

Stubhead	Column head	Spanner head [1]			Reading column head	Reading column head
		Coal	Coke	Subspanner head: Carloads more than other years		
				Wood		
CENTERHEAD	*Millions of dollars*	*Tons*	*Tons*	*Number*		
1 Lead or caption line [2] ...	[3] 900	150	191	246	1,987	Feb. 12, 1958..... Reading column.[4]
2 Wheat and other grains.	189	257	250	379	1,235	May 9, 1957 .....
3 Lumber and mill work..	326	382	177	584	1,742	Dec. 31, 1957.....
4 .....do.[5]	573	176	263	129	1,963	.....do..... Do.[6]
5 Total line.....	1,988	965	881	1,338	6,927	..... (7).
CENTERHEAD						
6 Lead or caption line	1,057	(8)	286	2,673	1,891	July 19, 1958.... Same reading column with a runover.
7 Mining equipment	321	156	112	1,114	3,821	May 3, 1958 ..... Do.
8 .....do.....	769					
9 (8).....	258	387	596	342	2,297	June 15, 1958 ... Reading column.
10 Total line.....	2,405	543	994	4,129	8,009	.....



2 READING COLUMNS

(Leader from top line)

1	Dairy products:								1
2	In ctns (pounds)	1,485,692	380	462	.....	3,624	Mar. 3, 1958.....	Reading column.	Reading column..... 2
3	In cans (pounds).....	263,491	.....	198	3,762	5,783	Dec. 17, 1957.....	Do.	.....do..... 3
4	Clay products (other than pottery, refractories) (boxes).	325,000	621	.....	4,111	1,926	Nov. 26, 1957....	Do.	.....do..... 4
5	Ferrous alloys (tons).....	163,381	556	276	1,985	112,812	.....do.....	Same reading column.	Same reading column.. 5

1 READING COLUMN

(Leader from bottom line)

1	A short line (boxes).....	13,092	748	365	2,421	986	7,654,000	5,137
2	A long, crowded line (thousands of tons)	5,692	345	721	2,679	3,542	962,111	4,728
3	A very long line that runs over (crates).....	386,591	475	582	13,563	12,297	82,129	18,591,763

[1] Reference number in boxheading.
[2] Reference number followed by leaders in stub or inside reading column.
[3] Reference number in figure column.
[4] Reference number in last or outside reading column.
[5] Reference number following "do" in inside reading column.
[6] Reference number following "Do" in last or outside column.
[7] Reference number standing alone in last or outside column, enclosed in parentheses followed by period "(7)," and quadded out to end of line.
[8] Reference number standing alone in figure column, enclosed in parentheses (8).
[9] Reference number standing alone in inside reading column, enclosed in parentheses (9), and leadered out to cast on right.

NOTE.—If no tracer-figure column is used on the left of table and the stub or reading column is set flush, "Do." will be capitalized and leadered out to cast on right.

Source: U.S. GPO (1984). *U.S. Government Printing Office Style Manual*, Washington, D.C.

Figure 15. Parts of a table.

Figure 16. Anatomy of a simple table.

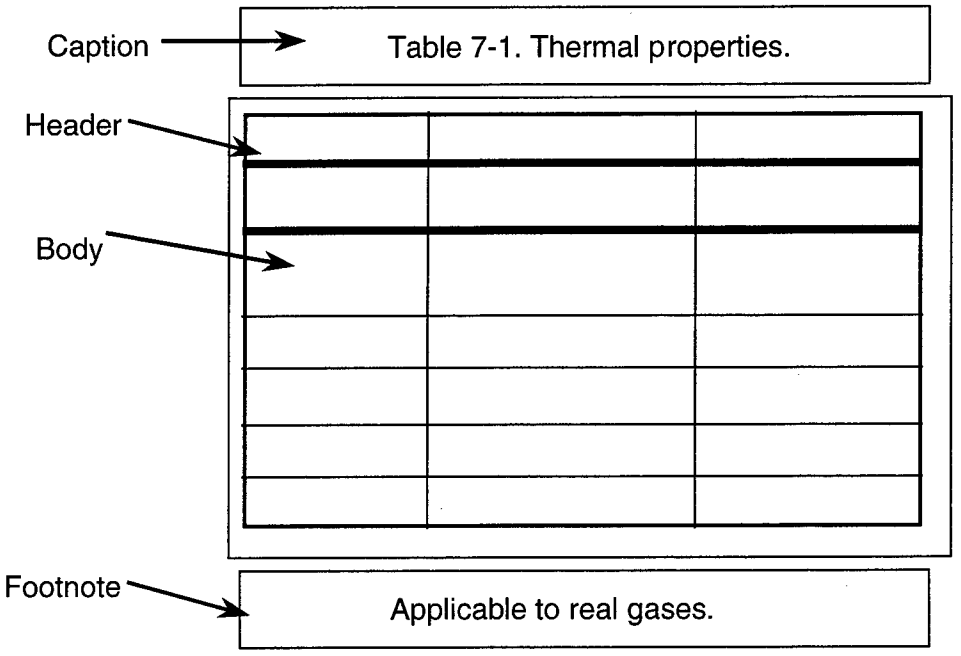
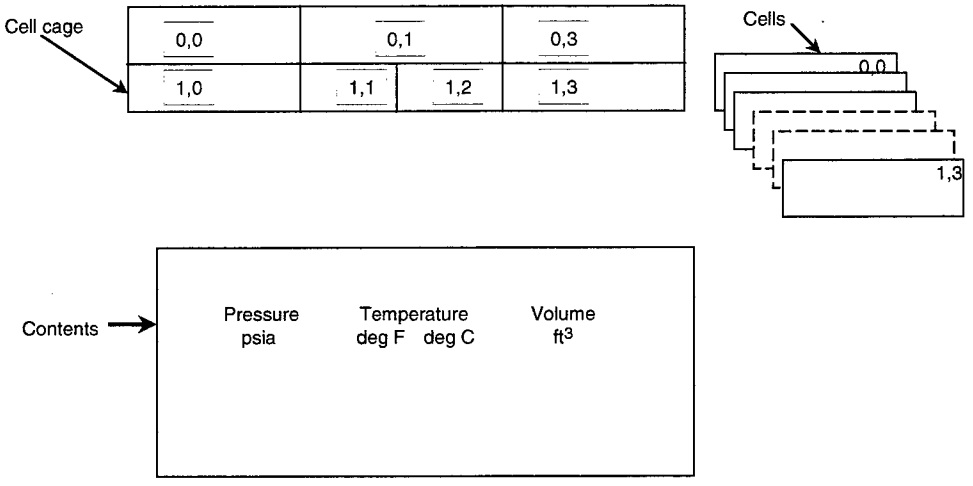


Figure 17. Table column header (boxhead) object.



The *table body* contains the data characterized in the column header. The elements in each column represent values pertaining to the corresponding column label. The elements along a row also usually form a consistent set; that is, they relate properties of a given state.* Consequently, each element in a table body is associated with its neighbors in a row and with its neighbors in a column.

*In some unusual tables, the columns are merely unassociated lists. Although such tables are the minority, they should be included in the analysis. One way to analyze such a column might be as one cell containing a list, rather than a set of cells. Natural Computing software should be able to reveal both relationships and lack of relationship.

An element in a table body can be a number, text, a graphic symbol, or any combination that makes sense to a user. Elements within a table can represent continuous states of a field (such as the thermal properties of a gas or the depth of a sea over a given area). Or the values in the table can be discrete and represent values only at distinct (finite) states. In the former, values can be interpolated, while interpolation is meaningless in the latter.

Table data body elements (fig. 18) are usually uniform, being either textual data or numbers. But it is not uncommon to see variations and omissions on this uniformity of data. Table elements carry blanks, dashed lines, a series of dots, remarks, and so forth.

The first column in many tables stands in a special relationship to the other columns; elements in the first column often characterize in some way the other elements in the rows. (In GPO terminology, the first column is called the "stub" column; see fig. 15.) This relationship may be similar to that between the column header and the table body; for some tables, the first column could act as the column header if the table were rotated 90°. I refer to such a column as a *row header*, to capture the parallel with the column header. (Although it is not at the head of the table in the sense of being at the top (as the column header is), its elements act as the "heads" of each row in a "command" sense.) It is also not unusual to find labeled subdivisions in a row header (in GPO terms, "centerheads" in the stub column). These labels correspond to spanner heads in the boxhead (see fig. 15), and reveal another way in which column headers and row headers can have parallel functions. In some types of tables, the first column information may be interpreted as the independent variable on which information in the other columns depends; for some relationships, it may be possible to "promote" another column to the first position and thus treat its information as independent. The various meanings that the first column can carry will require careful analysis.

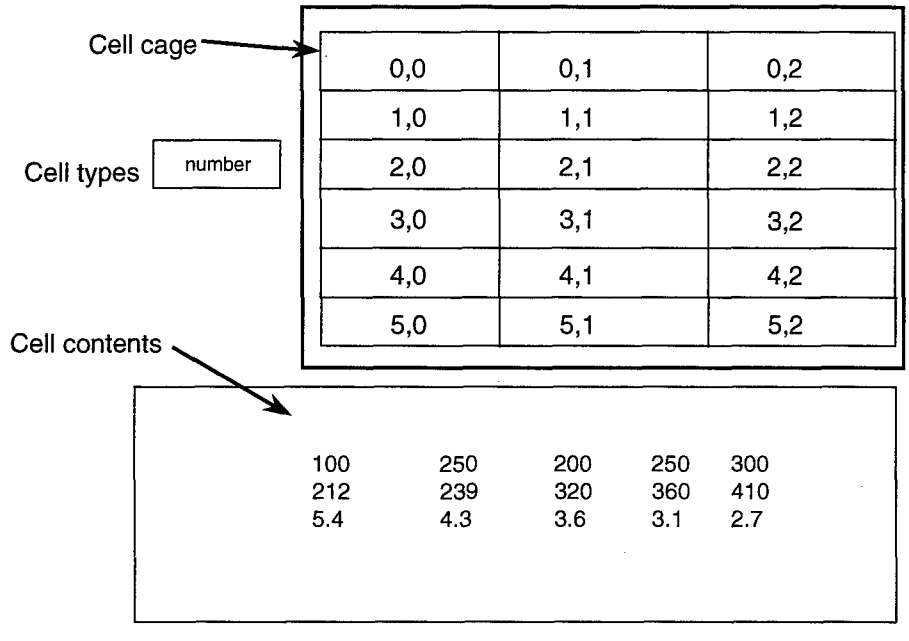


Figure 18. Table data body object.

A common optional feature of tables is *notes*. These qualification marks (head notes or footnotes) may be carried by some table elements. Some apply to the whole table, while others apply to specific rows, columns, or elements. Domain specialists use these marks and footnoted explanations to capture the vagaries of information. Users are expected to be cautious of the notes and be sure to apply necessary checks for maximum validity of the data and information.

9.1.2 *Capturing Meanings Expressed in Tables*

Real-world tables made up of the generic parts just described are used to express a variety of relationships. Over the years, people have developed table conventions that allow us to indicate these relationships, their applicability, and their limits. In Natural Computing, we will require software tools that are sufficiently rich to capture these meanings.

The steam tables alluded to earlier (sect. 5) demonstrate the generic structure and behavior of a table. Steam has a thermodynamic state, which is captured as a number of properties: pressure, temperature, specific volume, internal energy, enthalpy, and entropy. Given any two of these properties, the state is completely defined. Hence, the properties of steam or other gases can be represented by means of a two-dimensional table. No matter which gas the table represents, the structure and behavior of a table are generic from a software development perspective.

In some tables, the functional relationship holds only at the points given in the table, whereas in others, the data can be interpolated and/or extrapolated. Thus, Natural Computing will require the implementation of tables with differing behavior. Whether and how interpolation is done would be a property of the table: interpolations and extrapolations can be along the rows, columns, or both; also, this property would define which of several types of interpolations apply (arithmetic interpolation being the most common).

An interesting example is presented in figure 19, where cells in a column contain ranges of numbers (Shigley, 1977). When cells are filled with ranges instead of single numbers, we can (in essence) avoid the need for many if-then or switch-and-case statements in programming by the appropriate use of such table columns. In the example table, the first row of values can be used only if the size range is within 0.004 to 0.250 in. This demonstrates a clever means of representing an IF statement in a table! In some cases, as in the spring materials example shown in the figure, a table element may prescribe an allowable range for the rest of the relationship to hold. Ordinarily, however, an allowable point value is prescribed.

Table cells may contain the results of manipulation of other cells (e.g., the sum of values of a number of other designated cells), as in a spreadsheet. (Before the development of computer spreadsheets, people used such methods with simple tables and called them "worksheets.") In some instances, the value obtained from the table element is modified based on footnotes or other notes. Thus a table is not merely a lookup table; it also

Figure 19. A table example wherein cells in a column contain ranges of numbers.

Table 8-2 CONSTANTS FOR USE IN EQ. (8-10) TO ESTIMATE THE TENSILE STRENGTH OF SELECTED SPRING STEELS

Material	Size range, in	Size range, mm	Exponent, m	Constant, A	
				kpsi	MPa
Music wire[a]	0.004–0.250	0.10–6.5	0.146	196	2170
Oil-tempered wire[b]	0.020–0.500	0.50–12	0.186	149	1880
Hard-drawn wire[c]	0.028–0.500	0.70–12	0.192	136	1750
Chrome vanadium[d]	0.032–0.437	0.80–12	0.167	169	2000
Chrome silicon[e]	0.063–0.375	1.6–10	0.112	202	2000

[a] Surface is smooth, free from defects, and with a bright lustrous finish.
[b] Has a slight heat-treating scale which must be removed before plating.
[c] Surface is smooth and bright, with no visible marks.
[d] Aircraft-quality tempered wire; can also be obtained annealed.
[e] Tempered to Rockwell C49 but may also be obtained untempered.

Source: Joseph E. Shigley (1977). *Mechanical Engineering Design*, 3rd ed., McGraw-Hill Book Co., New York, NY.

informs us to *look out* for footnoted interpretations based on conditions. I call this quality the *responsibility* of an object during computations. In the Natural Computing approach to tables, it is important to ensure that a table object behaves responsibly; that is, it must account for the footnotes, the exceptions, and the units. The software tool developer, who develops table classes, should endow the tables with the correct behavior (for example, interpolation allowed or disallowed). The domain specialist, who programs a specific calculation and information in the domain, should select the table with the right behavior. If the table object behaves responsibly, the naive end user will have no problems with the tables and with the calculation capabilities of those tables.

Another aspect of a responsible table in my sense is the proper treatment of units. In the real world, variables do not come with consistent units across all interfaces and applications. For example, clothing manufacturers buy cloth in yards, but pattern cutters use inches for their measurements. We may enter a table with one set of units, while the table values are in a different set of units. The responsible user-friendly table should accommodate any consistent set of units.

Figure 20 shows a spectrum of tables (classes and categories). This display is called a profile representation of options (Warfield, 1994, and Warfield and Cardenas, 1994). It shows that a table header can have three options, a cell cage can have four options, and so on. As we pick each option and explore the various combinations, a large number of practical tables can be covered for representation in the Natural Computing system.

9.2 Operations

The extraction of information from a table is one of the simplest of operations possible with a table. People perform far more sophisticated operations on or with tables. For example, two tables can be added. A simple example is provided by tables 4 and 5.

Following the object-oriented programming style, we should be able to write

table 4 + table 5 = table 6,

and obtain the result in table 6.

The "+" operator can be overloaded so that it adds tables in a specified way. We obtain the total sales for both regions by adding the values in the two tables. Note that not all elements, but only the appropriate data elements (those in the sales column) are added. This addition can be repeated a number of times, so that we can get weekly results by adding 7 tables of daily data, or annual results by adding 12 monthly data tables. In a similar manner, we can define subtraction, scalar multiplication, and scalar division operations for tables.

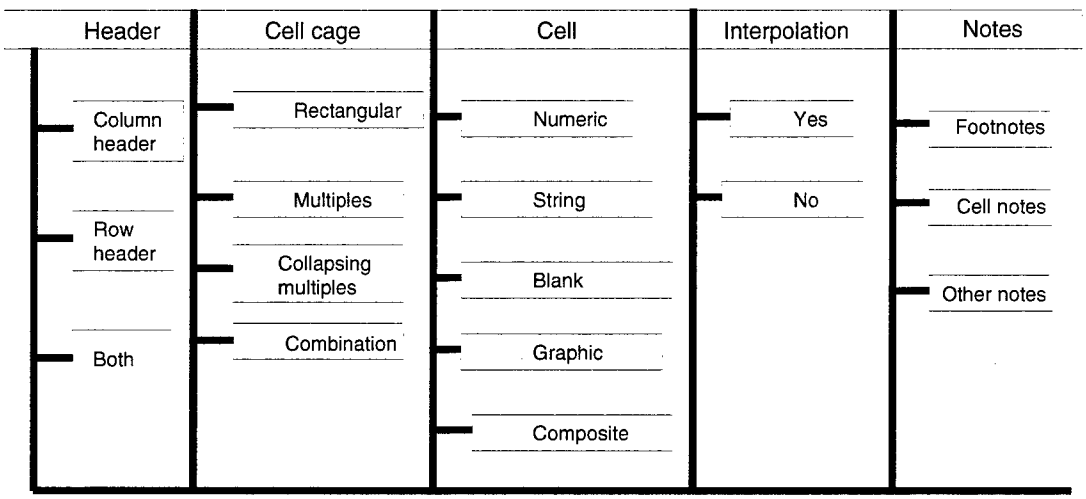


Figure 20. A spectrum of generable table types in profile representation.

Table 4. Sales in East Region.

Item	Sales ($)
Coffee	250
Donuts	890

Table 5. Sales in West Region.

Item	Sales ($)
Coffee	220
Donuts	1000

Table 6. Result table: sales in both regions.

Item	Sales ($)
Coffee	470
Donuts	1890

9.3 Representation of Structure

What representation would give software tables the functionality of real-world tables? Object-oriented programming techniques are the answer. A table class category can be defined, and specific table objects can be instances of classes from that category. Since tables come in a wide variety and serve a variety of functions, they call for a number of classes, which can be derived from base classes.

The table anatomy given in section 9.1.1 is a good starting point for designing a structure for the table. A table is first divided into several components (table title, column header, row header, table data body, and notes). The column header, row header, and table data body consist of several cells. These components themselves can be represented by composition classes (that is, a collection of other classes/objects) containing several cells. The cells in turn contain numbers, text strings, graphic symbols, and so forth. Therefore, the cells will be represented by several distinct types of classes. In a separate report (ARL-TR-2041), I discuss computer representations of tables in detail (Karamchetty, 2000a); here I give only a brief description.

Traditionally, cells in a table are represented as elements of a two-dimensional array. But such a representation inherently limits the table to a rectangular arrangement. It cannot handle tables where columns have subcolumns and where columns (or rows) combine. We can avoid this limitation by decoupling the table structure and the data (see fig. 21). Table structure (both headers and data body) is captured in a cell cage. The cells are numbered by their row and column indexes (0,0; 0,1; etc). An array of all cell locations and pointers to data is set up as shown in figure 21. We can define mathematical properties for the cell-cage structure. That is, the cell-cage location can be computed by means of the cell-cage array location, the cell-cage type, and cell-cage size. The array location and cell-cage location can yield adjacency lists that show which cells have neighborly relationships. The array element also points to the contents of the cell. This indirect pointer notation connects the cell-cage location and the contents. The separation of both the structure and the contents of a table is in harmony with the philosophy that operations on the table should be generic.

Table column headers (and row headers, where applicable) describe what the column (row) represents. Each *cell* of a header typically consists of a variable (and often a symbol for that variable) and a string descriptor.

In most science and engineering applications, symbols are used for variables, which may be explained on first use or in a list of notation (akin to the data dictionary in database systems). For example, in equations, tables, and graphs, one might use p as the variable standing for pressure. This definition would be included in the list of notation. Some books additionally carry a list of units, with entries such as "p, psia." But it is common to indicate the units on the list of notation, with entries such as "p, pressure (psia)." (Such an entry could correspond to a column header

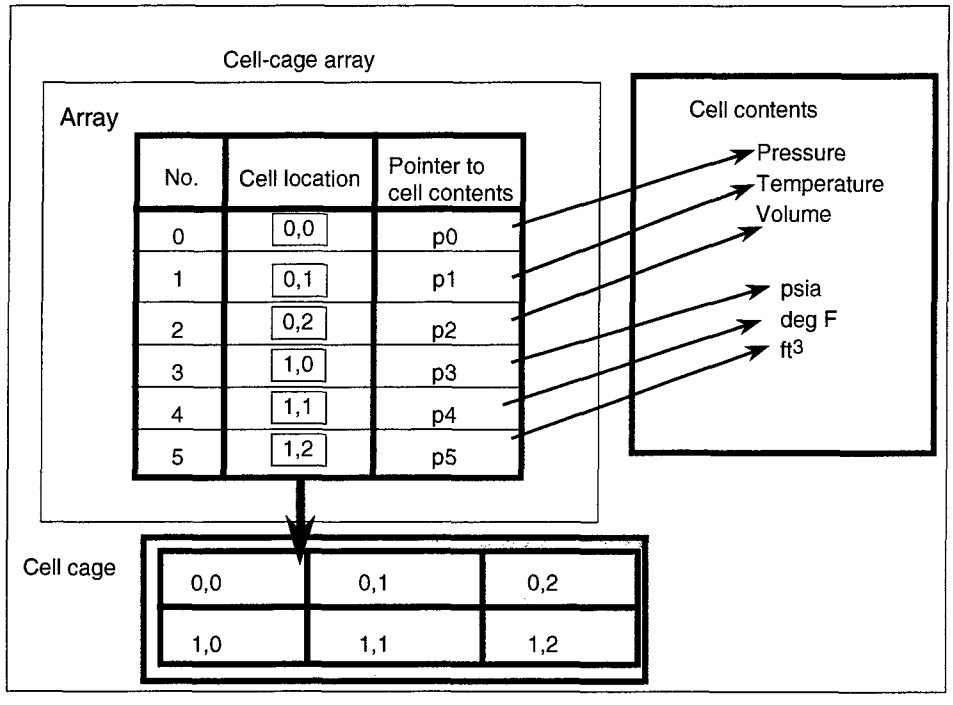


Figure 21. Cell-cage array connects cell locations and contents.

cell.) In a Natural Computing document, the list of notation will bring together all the variables in the document along with explanations. In fact, each table header cell pointer should point to a unique memory location where the variable and its notation and explanation are stored. In some cases, different variables are used for the same quantity in separate contexts; such situations can be handled separately.

9.4 Development, Choice, and Use

In the foregoing, I have identified a number of different types of tables. The most frequently used tables can be selected, programmed by a software developer, and provided in a tool box or Natural Computing environment for use by domain specialists, who will select a table type that best fits the needs of a particular application. Thus, an instance of a blank table is created. In the creating/editing mode, all data and information are entered into the table by the domain specialist, who is also responsible for choosing and filling in the footnotes and other notes as applicable. The table object also develops a number of behavioral characteristics for the domain specialist to review. For example, the domain specialist records the limit values (minimum, maximum, and singularities) of a characteristic. The table object would use these limits to flag an error message if a user tries unallowable values. This is another table feature that we want to capture so that our software can imitate human usage of tables; in real-world applications, domain specialists often provide such checks. Minimum and maximum values of a variable prevent extrapolation outside allowable bounds.

A table object will also have a set of input (or query) templates and output templates. Many tables describe functional relationships among

quantities (variables or characteristics) identified in the cells of the column header. Given one of them (and appropriate data), the others can be determined from the table. We can use this functional relationship in generating query templates. The user should be able to choose the appropriate input template, type in a value for the independent variable, submit it to the table object, and obtain the result in an output template. The input template also guides the user with the limits on the variable values. These guidance values in the templates protect a table from invalid or out-of-range queries. A table with input and output templates can be likened to a computer hardware component with its input and output sockets and pins. These input and output templates are very useful in connecting different functional objects into a procedure (procedures are described in sect. 12).

9.5 Search for Data

Since the cells in a table can be connected by adjacency lists and pointers to neighbors, and pointers lead to data values or contents, searching a table for result data items is very simple. A query consists of an independent-variable/value pair and a dependent variable. We wish to find the value corresponding to the dependent variable. We use the simple property of a table that two value cells bear the same neighborly or adjacency relationship as the corresponding two variable cells do. This relationship is depicted in figure 22.

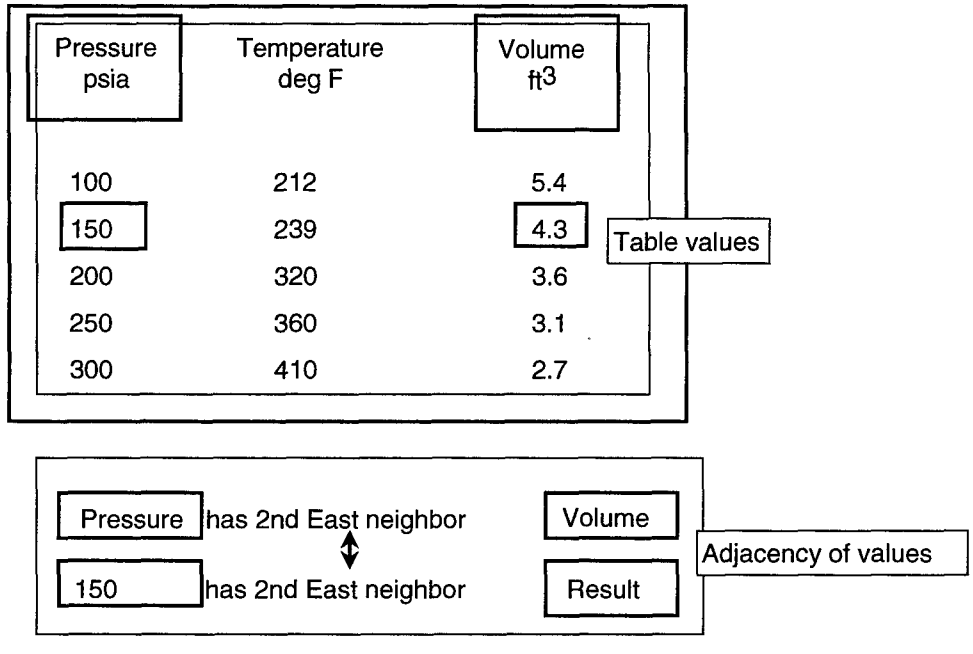


Figure 22. Adjacency property is used to get table values.

9.6 Visibility of Data

Educators criticize current software systems as black boxes: because the solution method is opaque to the user, the user learns nothing from the software. Even domain specialists do not understand what is in the code once their domain information is coded by a software developer. With traditional media (paper, calculator, and pencil), students continue to learn as they solve a number of problems. With traditional computer software, a student's learning has no correlation with the number of problems solved. In contrast, because tables in Natural Computing reveal themselves and show relationships between variables, a student can realize opportunities available and watch out for pitfalls in the problem domain represented by each table.

9.7 Table Data Storage

In Natural Computing, data in table components are stored in tagged or named and connected arrays. A domain specialist would choose the type of table needed for a specific application. Generic table structures and methods would be automatically available for the application. The specialist would also specify the size of the table.

9.8 Display

As pointed out in section 8, current software systems have addressed the display of tables extensively. Display is an important aspect of Natural Computing, since domain specialists and users interact with displayed tables. Again, display functions and methods are developed for generic tables. The end users are given greater flexibility to choose the displays that most suit their needs and comfort level. Natural Computing should follow the display practices that are commonly found in books and currently popular software, since these represent the most natural way for users to use and understand tables. However, three types of displays are needed: (1) for tables embedded in text, (2) for activated tables for interactive calculations, and (3) for tables 40

to create procedures.

9.9 Testing a Table in Isolation

Testing is a key task in software development. Capturing an application domain in software is equally critical. By isolating a table and testing it for a variety of inputs, together with the built-in justifications, limits, behaviors, and responsibilities prescribed for a table class, a Natural Computing programmer can go a long way toward eliminating bugs in software that uses tables. Since the filters on the table will allow only preapproved types and ranges of values, isolated testing can come very near guaranteeing the software and the domain information.

9.10 Growth of a System

Two types of system growth can be anticipated. The first is domain growth. A given domain will incorporate more complex problems, and other complex problems will cover a wide variety of domains. As domain coverage grows, tables will need to be represented with more complex features. As new table features (either structure or behavior) are encountered (or invented), software developers will play a primary role in developing those extended features. This second kind of growth will result in new software tools and environments or new versions of software.

10. Graphs

In section 9, I focused on describing table objects to demonstrate the importance of using natural objects, as well as showing methods to implement them. However, in addition to tables, Natural Computing consists of graphs, equations, and diagrams. The philosophy for representing these objects is similar to that used for table objects. In this section, I briefly describe the representation of graph objects.

10.1 State of Art in Graph Representation

The last decade of computer development has seen an explosion of graphics-based programming. Currently, powerful picture and image presentation and processing software systems are commercially available. Many programs generate and *output* excellent graphs, curves, or charts. Figure 4 is an example of an excellent-looking and complex graph generated by Mathematica® for a user to see but to do no other computation.

This class of programs generates graphs as *ends* (in the words of Coad and Yourdon, 1991) in themselves. Currently, no programs exist that use graphs as the *means* for calculations. In paper-based calculations, users use graphs to observe trends and discern the behavior of variables. Most notably, they use graphs to obtain resultant values. Using graphs in calculation helps people understand the "physics" of a problem. Take the example of the pipe friction coefficient graph in figure 23 (Karassik, 1976). With sophisticated equation-fitting programs, it is possible to develop a highly complex set of equations to replace this graph. But in the process, the intuition into the problem is completely lost. A hydraulics specialist and an end user will notice three distinct regions from this graph. The first one is the laminar flow region, where the friction coefficient is a function of Reynolds' number and is not a function of surface roughness. Then there is a transition region where the pipe friction is not well defined. The third region is the turbulence region, where the pipe friction is strongly influenced by surface roughness. For each pipe roughness value, the curves generally tend to become flat after a certain Reynolds' number value. No equation can represent the functional relationship depicted by the graph in figure 23 and be intuitive enough that a domain specialist or user can notice that he or she is operating in a distinct region of behavior of the fluid (Streeter, 1971). On the other hand, the graph teaches or reminds the user that two distinctly different variables influence the results in the regions. The visibility of the underlying processes in computing is the most important reason to discard black-box programming and use visible and intuitive programming methods. Natural Computing graphs are essential to maintaining this visibility and intuitive usefulness.

Many of the points made in section 9 about tables also apply to graphs and other Natural Computing objects. In this section, I briefly describe some of the features specific to graphs.

*Such as Theorist and TableCurve (described in Douglas A. Smith and James P. Adams, *Scientific Computing and Automation*, July 1993, 27–28).

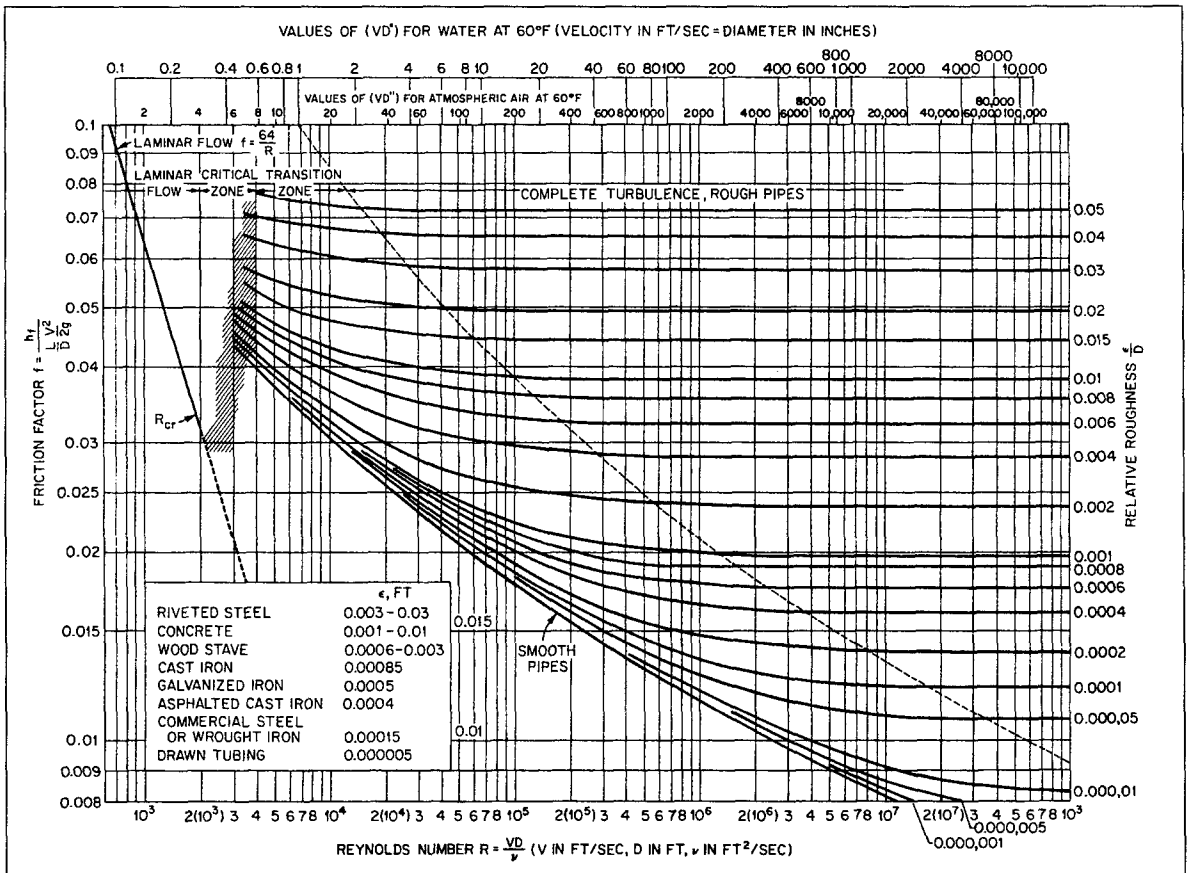


Fig. 28 Moody diagram. (V. L. Streeter, "Fluid Mechanics," 5th ed. Copyright 1971 by McGraw-Hill Book Company, New York)

Source: Igor J. Karassik, William C. Krutzsch, Warren H. Fraser, and Joseph P. Messina, eds. (1976). *Pump Handbook*, McGraw-Hill Book Co., New York, NY.

Figure 23. Example graph showing how a graphical relationship provides visibility to data and information.

10.2 Natural Computing Graph Representation

One represents a graph object by declaring, defining, and implementing a graph class. For computer internal storage purposes, graphs can be represented either by a table of x and y coordinates or by an equation. In the former case, a graph class will have a two-dimensional array of data members containing the x and y coordinates. The number of sets of points depends on the complexity of the graph and the desired accuracy. If there is more than one line or curve in the graph, a multidimensional array can be used and a parameter value is associated with each line or curve. Consequently, a graph class, in turn, will use an equation object as a member or a table object as a member to represent the data for the graph.

Graphs also have captions. The captions are listed in a table of contents to indicate all graphs available in a document. The other salient members of a graph class are the x and y axes, labels for x and y axes, and other notes.

Like tables, some graphs can be interpolated along a curve, among a set of curves, or both, while some other graphs cannot be interpolated. This interpolation characteristic is an attribute of a graph class. The information needed to display a graph (picture size, grid lines, line styles, and so forth) is captured as part of the display data members of a graph. It is interesting to note two points: (1) the display-related information is not needed for calculations of results using graphs, and (2) calculations are essentially done as if the data were in a table or an equation. Graphs should carry a cursor whose shape can be changed at the user's discretion. A full-size crosshair could be moved along the graph that highlights the point on the graph that is being examined. In place of, or besides a cursor, a property window could be displayed, indicating numerical values at the point of examination. As a user scans a graph, the cursor or crosshairs should move over the graph, and the property window displays the numerical values of the variables at the point on the graph.

On hard copies, the accuracy obtainable from graphs is limited by the paper size. On the computer, zooming in allows the accuracy of graphs to be improved, if detailed data are available. However, the use of a property window can give us highly accurate results up to several significant digits, perhaps making zooming unnecessary. (Excessive zooming could actually lead to a temporary loss of the visibility of the overall trend of results.)

Graphs would carry input and output boxes so that the user can type in input values and see the results from the graph in the output box. Both types of boxes would carry names of the variables and their units. Graphs too would have filters and justifiers to check the consistency of data and units. Graphs would also have lower and upper bounds, as well as singularities in between. With these facilities, graphs could be seen, interactively used, and connected into procedures. In many ways, graph objects will have characteristic behaviors similar to those of tables (graphs are the subject of a separate report (Karamchetty, 2000b)).

The example of steam tables is again applicable. Historically, steam engineers used the Mollier chart, which represented the properties of steam graphically. One could quickly perform or do steam power plant calculations by laying out the thermal processes on the Mollier chart. One could visually represent the complete steam cycle by marking the various state points on the chart.

The old paper Mollier charts suffered from inaccuracies. Replacing them with Natural Computing graphs in electronic media and using property windows could improve accuracy while retaining their visual benefits. Steam cycles could be laid out on these charts with great speed. A Mollier chart represents the properties of liquid water, water and steam mixture, and superheated steam. Various temperature, pressure, steam quality (dryness), and specific volume contours are drawn on the chart. A line is also shown that presents the states that separate the liquid and mixture states on the one side and the vapor and mixture states on the other. The contours are completely interpolatable for all properties.

Steam engineers find the chart highly useful in ensuring that various dryness qualities are adhered to in designing equipment. Although computer programs are now available to calculate steam cycles, none of them can match the visibility and simplicity of a Mollier chart. With Natural Computing, one can restore the visibility while maintaining the accuracy and speed of the computing methods. Additionally, one can lay problems out on the chart on the fly, while preserving the speed, accuracy, and efficiency of computer calculations.

11. Equations

Equations are the last of the three functional representations. I do not propose to describe this feature in any great detail, as a number of current software systems allow equations to be typed in and used as functional relationships. But the use of equations along with graphs and tables in Natural Computing is very powerful. Moreover, the representation and use of equations follow the philosophy and methods described in the foregoing sections.

Equations are developed as a category of classes. An equation class consists of member data representing the operators and operands in a given sequence. Their relationship is captured in the form of a tree structure.

A noteworthy feature of equation classes in Natural Computing is that they have a justifier function that protects an equation object from mixing quantities of dissimilar units. For example, the units justifier evaluates the units of each expression and flags inconsistencies in the units. For example, the units of all operands of the operators =, +, and – should be the same.

Like tables and graphs, equation classes also carry input and output boxes so that the user can type in input values and see the results from the graph in the output box. Both types of boxes carry the names of the variables and their units. Equations are identified by an equation number and caption. These equation numbers and captions can be grouped into a list of equation captions for indexing. (Although the use of equation captions is not common in printed media, they would clearly be useful for portable electronic documents.) Figure 24 shows an example of an equation with input and output templates for inserting inputs and obtaining result values.

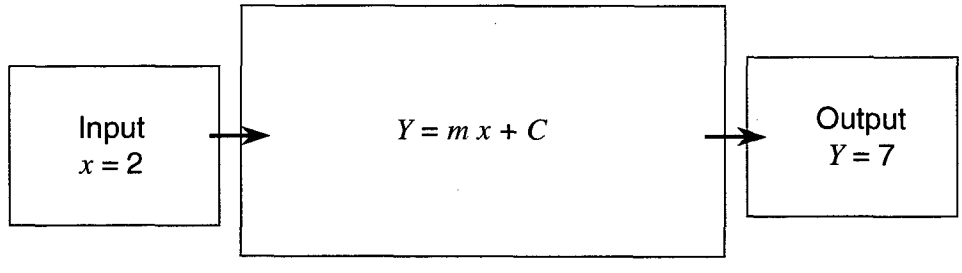


Figure 24. An example showing interactive use of an equation.

12. Procedures

In paper-based calculations, people develop procedures by combining several objects (tables, graphs, and equations). The output of one object is used as input to another object, and so on in some sequence. People use each of the objects, note intermediate results on a scratch pad, carry them into the next object, and so on.

In a predefined procedure, this chaining is done at the variable level. To facilitate this process, in Natural Computing, I define a new object called a *pipe*. In its simplest form, a pipe has a donor and a donee (a recipient). A donor is a variable that provides its value to the pipe. The pipe carries that value to the donee, which is a variable in another (downstream) object. In more complex pipes, a pipe can have several pairs of donors and donees.

One can set up procedures by chaining several equations, tables, graphs, and other procedures to compute a complex calculation repeatedly. In that sense, a procedure is like a computer program. But a procedure still retains a high level of visibility, since the user can open up a procedure and inspect its component objects.

A large procedure can be built from a number of small procedures. By testing and verifying each procedure in isolation, one can assure high quality for the composite procedure. A domain specialist can reuse and rearrange the components of a procedure and develop a new procedure. Figure 25 shows a procedure that consists of several graphs and tables connected. This procedure can be inserted into a program as step *k*. When a procedure is opened up, the details of it become visible.

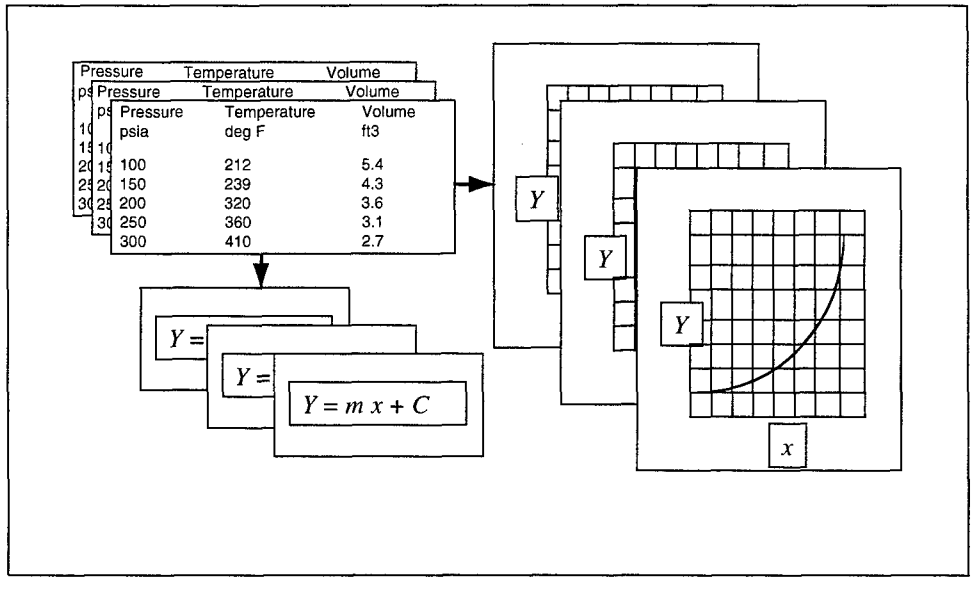


Figure 25. A Natural Computing procedure example.

13. Pictures

Paper-based computing uses sketches, drawings, and schematics to show the relative positions of variables and properties. For example, a sketch or a drawing may show a number of dimensions of a component or an assembly. Some of the dimensions are used in equations, tables, and graphs. One can set up a procedure in which the dimension placed on a sketch or a drawing can be piped into a procedure and made part of that procedure. Or in reverse, a calculation can generate a size, which can be piped into a drawing. Figure 26 shows a sketch from which values of variables can be connected to an equation.

Schematic diagrams are extensively used in engineering. A schematic diagram shows how various components are connected and how a flowing fluid, for example, changes its state. Figure 11 showed a schematic diagram of a thermal power plant. Water or steam flows through pipes and components, and during the course of its flow, steam changes its state. There are equations, tables, and/or graphs that relate various properties of these states. Schematics add another dimension to the visibility and understanding of a problem.

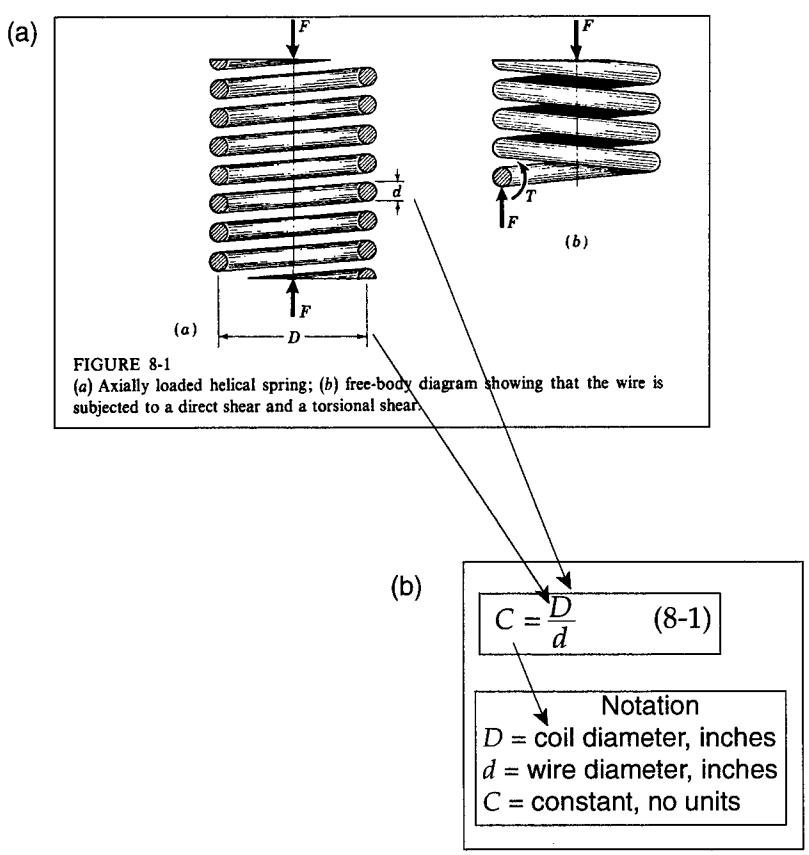


Figure 26.
Dimensions from (a) a sketch used in (b) an equation.

14. Text

Natural Computing calculation features are embedded in text as in a document or book. Text along with tables, graphs, and equations is displayed for a reader to peruse. Two other modes are available: use within a document, or reuse in other documents. Any time the end user wishes to do a simple calculation using any of these objects embedded in text, that object can be activated. Appropriate inputs into the objects will produce resultant values. After using those objects, the end user can revert to reading the text by closing those objects. This is the within-document mode. In the reuse mode, an end user can develop new procedures by copying objects from documents and connecting them. These procedures can be saved for future use. Figure 27 shows a procedure embedded in text.

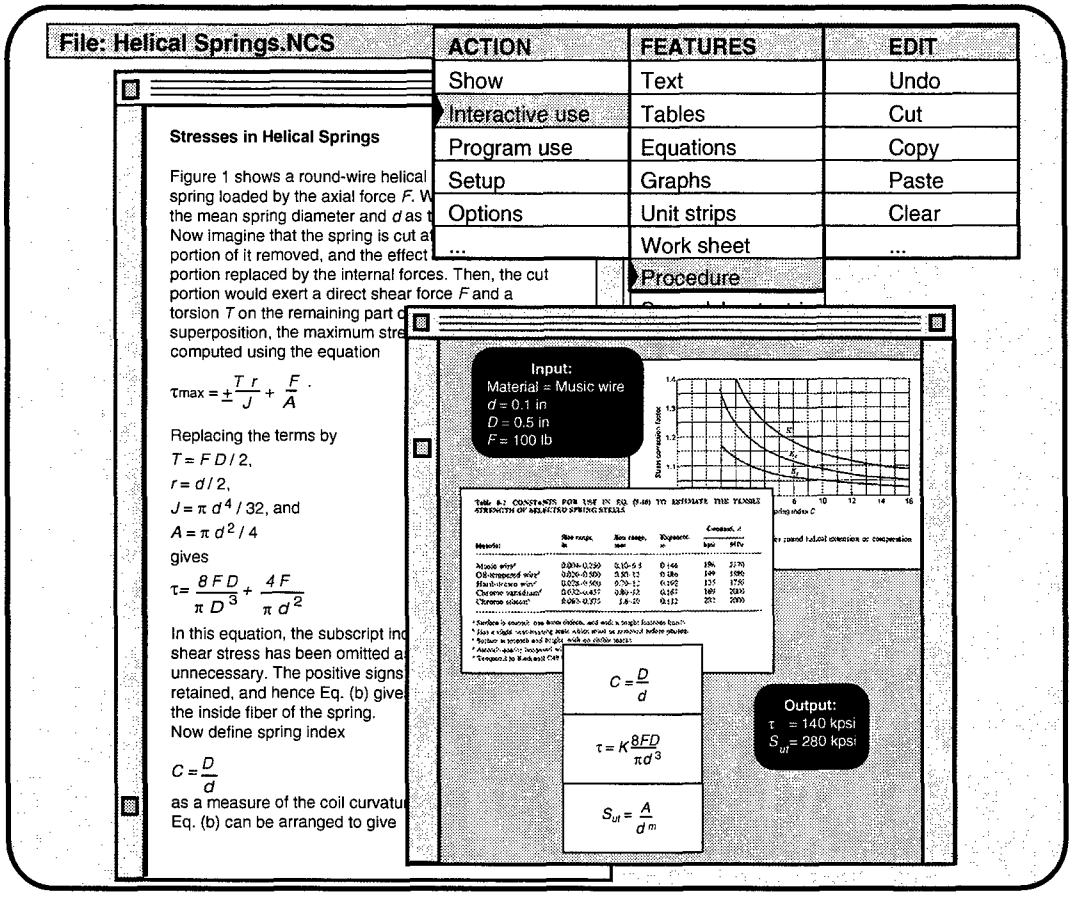


Figure 27. Natural computing text with procedure.

15. Conclusions

Traditional programming or software development focused on making it easy for a software developer while he or she tried to understand a domain and worked in it. Database systems aimed at achieving mathematical correctness and elegance (Date, 1995) versus emulating real-world tables. Software systems, such as word processors and computer-aided drafting systems, have broken that tradition and provided domain users with generic software for use with no further assistance from software developers.

Scientists and engineers use information from reference books, textbooks, and journals. Reference information is useful over long time periods, while textbook information is more recent. Journal information relates to the latest inventions or discoveries. With Natural Computing documents, after reading about an interesting new scientific discovery in a journal, a scientist can readily combine it with existing information in textbooks or reference books, since this process of combining information is extremely simple. The scientist can copy the Natural Computing feature (equation, graph, table, procedure, or picture) and combine it with an existing procedure. Presto! A new algorithm or procedure is instantly developed for further use. The scientist can instantaneously communicate it to others interested in the new procedure. This is what I mean by asserting that Natural Computing allows for the creation and communication of new knowledge.

Natural Computing focuses on developing generic calculation software for use by domain specialists in science, engineering, and other fields that depend on calculation. With the Natural Computing environment and tools, domain specialists could work with tables, graphs, equations, procedures, pictures, and text. Among the benefits of this approach are economical and affordable generation of applications, high quality of software and applications, high growth potential for applications, high reuse of applications, ready availability of high-utility information in electronic media, and rapid transmission of domain information.

Acknowledgments

The following Companies have given permission to use their copyrighted material and the author's grateful to them:

- McGraw-Hill Book Co., New York, NY,
- The Washington Post, Washington, DC, and
- John Wiley and Sons, Inc., New York, NY.

References

Brooks, Frederick P., Jr. (1987). "No Silver Bullet: Essence and Accident of Software Engineering," *Computer*, **20**, No. 4 (April), 10–19.

Coad, Peter, and Edward Yourdon (1991). *Object-Oriented Analysis*, 2nd ed., Yourdon Press Computing Series, Prentice Hall, Englewood Cliff, NJ.

Date, C. J. (1995). *An Introduction to Database Systems*, 6th ed., Addison-Wesley Publishing Co., Reading, MA.

Karamchetty, Som D. (1997). "Natural Computing," U.S. Patent Number 5,680,557 (October 21).

Karamchetty, Som D. (2000a). *Natural Computing: Analysis of Tables for Computer Representation*, U.S. Army Research Laboratory, ARL-TR-2041.

Karamchetty, Som D. (2000b). *Natural Computing: Analysis of Graphs for Computer Representation*, U.S. Army Research Laboratory, ARL-TR-2042.

Karassik, Igor J., William C. Krutzsch, Warren H. Fraser, and Joseph P. Messina, eds. (1976). *Pump Handbook*, McGraw-Hill Book Co., New York, NY.

Keenan, Joseph H., Frederick G. Keyes, Philip G. Hill, and Joan G. Moore (1969). *Steam Tables: Thermodynamic Properties of Water Including Vapor, Liquid, and Solid Phases*, John Wiley and Sons, Inc., New York, NY.

Lemay, Laura (1996). *Teach Yourself Web Publishing with HTML 3.0 in a Week*, 2nd ed., Sams.Net, Publishing, Indianapolis, IN.

Morris, Mary E. S. (1996). *HTML for Fun and Profit*, Sunsoft Press.

Shaler, Sally, and Stephen J. Mellor (1988). *Object Oriented Systems Analysis—Modeling the World in Data*, Yourdon Press Computing Series, Prentice Hall, Englewood Cliffs, NJ.

Shigley, Joseph E. (1977). *Mechanical Engineering Design*, 3rd ed., McGraw-Hill Book Co., New York, NY.

Streeter, V. L. (1971). *Fluid Mechanics*, 5th ed., McGraw-Hill Book Company, New York, NY.

U.S. Government Printing Office (GPO) (1984). *U.S. Government Printing Office Style Manual* (March).

Van Wylen, Gordon J., and Richard E. Sonntag (1965). *Fundamentals of Classical Thermodynamics*, John Wiley and Sons, New York, NY.

Warfield, J. N. (1994). *A Science of Generic Design: Managing Complexity Through System Design*, 2nd ed., Iowa State University Press, Ames, IA.

Warfield, J. N., and A. R. Cardenas (1994). *A Handbook of Interactive Management*, 2nd ed., Iowa State University Press, Ames, IA.

The Washington Post (1996) Monday, May 27, p C6.

Wolfram, Stephen (1991). *Mathematica, A System for Doing Mathematics by Computer*, 2nd ed., Addison-Wesley Publishing Co., Reading, MA.

Distribution

Admnstr
Defns Techl Info Ctr
Attn DTIC-OCP
8725 John J Kingman Rd Ste 0944
FT Belvoir VA 22060-6218

Ofc of the Secy of Defns
Attn ODDRE (R&AT)
The Pentagon
Washington DC 20301-3080

Ofc of the Secy of Defns
Attn OUSD(A&T)/ODDR&E(R) R J Trew
3080 Defense Pentagon
Washington DC 20301-7100

AMCOM MRDEC
Attn AMSMI-RD W C McCorkle
Redstone Arsenal AL 35898-5240

CECOM
Attn PM GPS COL S Young
FT Monmouth NJ 07703

Dir for MANPRINT
Ofc of the Deputy Chief of Staff for Prsnnl
Attn J Hiller
The Pentagon Rm 2C733
Washington DC 20301-0300

TECOM
Attn AMSTE-CL
Aberdeen Proving Ground MD 21005-5057

US Army ARDEC
Attn AMSTA-AR-TD M Fisette
Bldg 1
Picatinny Arsenal NJ 07806-5000

US Army Info Sys Engrg Cmnd
Attn ASQB-OTD F Jenia
FT Huachuca AZ 85613-5300

US Army Natick RDEC
Acting Techl Dir
Attn SSCNC-T P Brandler
Natick MA 01760-5002

US Army Simulation, Train, & Instrmntn Cmnd
Attn J Stahl
12350 Research Parkway
Orlando FL 32826-3726

US Army Soldier & Biol Chem Cmnd
Dir of Rsrch & Techlgy Dirctrt
Attn SMCCR-RS I G Resnick
Aberdeen Proving Ground MD 21010-5423

US Army Tank-Automtv Cmnd Rsrch, Dev, & Engrg Ctr
Attn AMSTA-TR J Chapin
Warren MI 48397-5000

US Army Train & Doctrine Cmnd
Battle Lab Integration & Techl Dirctrt
Attn ATCD-B J A Klevecz
FT Monroe VA 23651-5850

US Military Academy
Mathematical Sci Ctr of Excellence
Attn MDN-A LTC M D Phillips
Dept of Mathematical Sci Thayer Hall
West Point NY 10996-1786

Nav Surface Warfare CtrA
Attn Code B07 J Pennella
17320 Dahlgren Rd Bldg 1470 Rm 1101
Dahlgren VA 22448-5100

DARPA
Attn S Welby
3701 N Fairfax Dr
Arlington VA 22203-1714

Palisades Inst for Rsrch Svc Inc
Attn E Carr
1745 Jefferson Davis Hwy Ste 500
Arlington VA 22202-3402

NASA Langley Rsrch Ctr
Vehicle Techlgy Ctr
Attn AMSRL-VT W Elber
Hampton VA 23681-0001

Distribution (cont'd)

US Army Rsrch Lab
Attn AMSRL-WM I May
Aberdeen Proving Ground MD 21005-5000

US Army Rsrch Lab
Attn AMSRL-RO-D C Chang
Attn AMSRL-RO-EN W Bach
PO Box 12211
Research Triangle Park NC 27709

US Army Rsrch Lab
Attn AMSRL-CI N Radhakrishnan
Aberdeen Proving Ground MD 21005-5067

US Army Rsrch Lab
Attn AMSRL-HR R L Keesee
Aberdeen Proving Ground MD 21005-5425

US Army Rsrch Lab
Attn AMSRL-VP R Bill
21000 Brookpark Rd
Cleveland OH 44135-3191

US Army Rsrch Lab
Attn AMSRL-SL J Wade
White Sands Missile Range NM 88002

US Army Rsrch Lab
Attn AMSRL-DD J Miller
Attn AMSRL-CI-AI-A Mail & Records Mgmt
Attn AMSRL-CI-AP Techl Pub (3 copies)
Attn AMSRL-CI-LL Techl Lib (3 copies)
Attn AMSRL-IS J D Gantt
Attn AMSRL-IS-CB L Tokarcik
Attn AMSRL-IS-CD P Jones
Attn AMSRL-IS-CI B Broome
Attn AMSRL-IS-CS G Racine
Attn AMSRL-IS-D COL M R Kindl
Attn AMSRL-IS-D P Emmerman
Attn AMSRL-IS-D R Slife
Attn AMSRL-IS-E D Brown
Attn AMSRL-IS-TA J Gowens
Attn AMSRL-SE J Pellegrino
Attn AMSRL-SE-EP S Karamchetty
(30 copies)
Attn AMSRL-ST C I Chang
Adelphi MD 20783-1197

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 2000	Final, 1995–1997

4. TITLE AND SUBTITLE Natural Computing: Its Impact on Software Development

5. FUNDING NUMBERS
DA PR: N/A
PE: N/A

6. AUTHOR(S) Som Karamchetty

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
Attn: AMSRL-IS-C email: skaramch@arl.mil
2800 Powder Mill Road
Adelphi, MD 20783-1197

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-2040

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
2800 Powder Mill Road
Adelphi, MD 20783-1197

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
ARL PR: N/A
AMS code: N/A

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
Many software engineering problems stem, in part, from the need for software designers to understand specialized knowledge domains. Current computer software systems are not capable of representing familiar calculation features such as equations, tables, graphs, procedures, and pictures so that these features assist humans to perform calculations in a natural, intuitive way. This report explains the need for these features to present users with "natural" ways of doing calculations—that is, ways analogous to the paper-based techniques used in the absence of computers. Features presented in this way would make computing more transparent and intuitive. In the *Natural Computing* approach proposed in this report, software tools are first developed and then given to domain specialists to use in their calculation methods, knowledge, and data. As domain knowledge changes and grows, and/or new calculation methods are needed, software developers can add new methods and procedures to the existing methods (or delete old ones) and develop successively enhanced versions of application software for use by both specialists and naive end users. Domain information and knowledge can be captured in electronic books and communicated electronically for further expeditious use. Natural Computing eases application system development and accelerates domain knowledge dissemination, leading to quicker development of further knowledge.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Software engineering, tables, graphs, evaluations, object-oriented programming	55
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

TO THE RECEPIENT OF THE COPY OF THE ARL-TR-2040 SENT ON 2 MARCH 2000:

SUBJECT: ARL-TR-2040, NATURAL COMPUTING: ITS IMPACT ON SOFTWARE DEVELOPMENT BY SOM KARAMCHETTY

We regret that the copy of the report sent to you on 2 March 2000, contained figures that were not legible in certain copies. Please discard those low quality copies and replace them with the newly printed copy enclosed here.

We regret any inconvenience caused to you in this matter.

Army Research Laboratory